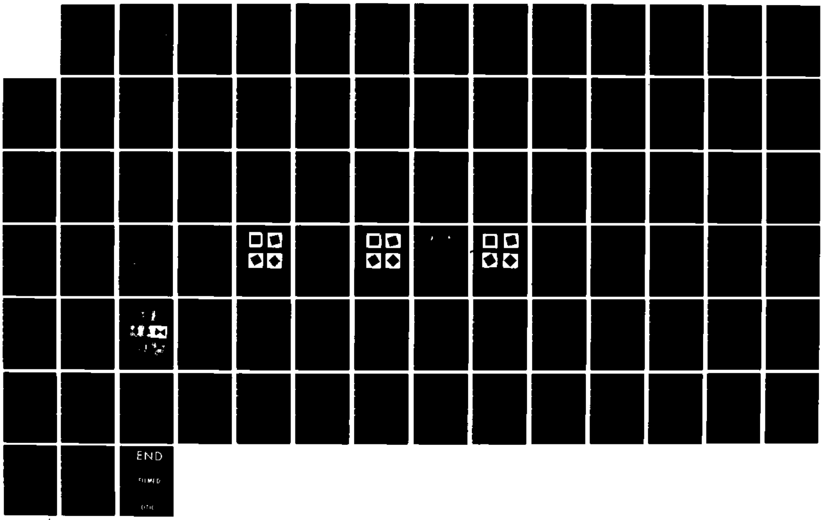
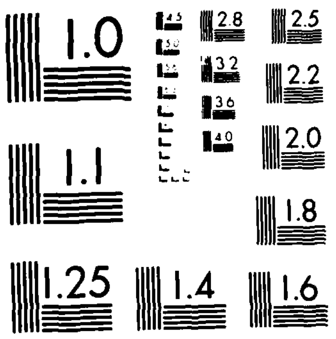
AD-A153 699 THREE-DIMENSIONAL DATA DISPLAY ON THE TWO-DIMENSIONAL SCREEN(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA H S HWANG DEC 84 1/1

UNCLASSIFIED F/G 9/3 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

②

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THREE-DIMENSIONAL DATA DISPLAY ON THE TWO-DIMENSIONAL SCREEN

by

Ho Sang Hwang

December 1984

Thesis Advisor: Chin-Hwa Lee

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A153699	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Three-Dimensional Data Display on the Two-Dimensional Screen		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December 1984
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ho Sang Hwang		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE December 1984
		13. NUMBER OF PAGES 83
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image Rotation, Image Projection, Dissection, Dissolution, Linear Interpolation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The difficulties encountered when implementing orthographic reprojection of 3-dimensional image data onto a 2-dimensional screen are considerable. They arise principally because of the size of data being manipulated and the tendency for underlying or overlying structures to obscure a clear view of the desired image. In this work, implementation was performed with an orthographic reprojection technique and many heuristic approaches were used to resolve some of the related problems.

Several coordinate rotating algorithms were tested in this work. Among them the Precalculation and Indexing method proved to be the most efficient algorithm.

Due to the disparity of the viewing-coordinate grids and the voxels of the volume data after rotation, 3-dimensional interpolation is required for appling the reprojection technique. Several methods implementing linear interpolation have been tried. Interpolation with a cone-shape kernel is the most appropriate method in the 2D situations and can be easily extended to a sphere-shape kernel in 3D situations.

The orthographic reprojection method includes a single plane dissection capability. Results on an artificial test data are collected using the above algorithms.
Several resulting images are included as well as the associated PASCAL programs.

Three-Dimensional Data Display on the Two-Dimensional Screen

by

Hwang, Ho-Sang
Lieutenant Colonel, Republic of Korea Air Force
B.S., Republic of Korea Air Force Academy, 1973

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1984

Author: Hwang Hosang
Hwang, Ho-Sang

Approved by: [illegible]
Chin-Hwa Lee, Thesis Advisor

Alex Gerba Jr.
Alex Gerba, Jr., Second Reader

[illegible]
Harriett B. Rigas, Chairman,
Department of Electrical and Computer Engineering

[illegible]
J. N. Dyer,
Dean of Science and Engineering

ABSTRACT

The difficulties encountered when implementating orthographic reprojection of 3-dimensional image data onto a 2-dimensional screen are considerable. They arise principally because of the size of data being manipulated and the tendency for underlying or overlying structures to obscure a clear view of the desired image. In this work, implementation was performed with an orthographic reprojection technique and many heuristic approaches were used to resolve some of the related problems.

Several coordinate rotating algorithms were tested in this work. Among them the Precalculation and Indexing method proved to be the most efficient algorithm.

Due to the disparity of the viewing-coordinate grids and the voxels of the volume data after rotation, 3-dimensional interpolation is required for appling the reprojection technique. Several methods implementing linear interpolation have been tried. Interpolation with a cone-shape kernel is the most appropriate method in the 2D situations and can be easily extended to a sphere-shape kernel in 3D situations.

The orthographic reprojection method includes a single plane dissection capability. Results on an artificial test data are collected using the above algorithms.

Several resulting images are included as well as the associated PASCAL programs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I wish to gratefully acknowledge my thesis advisor, Professor Chin-Hwa Lee, who provided help and assistance in the completion of this thesis.

I also would like to express my gratitude to Professor Alex Gerba,Jr. for his support.

I. INTRODUCTION

A. GENERAL

The representation of 3-dimensional(3D) objects on a 2-dimensional(2D) screen presents an interesting problem in computer science and computer engineering. The application of this technology is common in areas such as computer graphics, CAD/CAE, and medical imaging. The study here will be concerned with a more specialized area of showing 3D objects on 2-dimensional (2D) screen. A variety of techniques have been developed for this purpose.

A discrete set of 3D volume data consists of a collection of rectangular parallelepipeds. These are referred to as volume elements, or more commonly voxels. Each voxel has an associated value which represents the average intensity of a parallelepiped referred to as the density value. The density value is an integer from a finite set ($D_{min} \ldots D_{max}$). If $D_{min}=0$ and $D_{max}=1$, we say that the image is a binary image.

It is useful to classify 3D display techniques into different categories. [Ref. 1]. The main criterion used to classify them is whether the technique displays a whole scene or a only cross-section of a selected plane in the 3D volume data.

WHOLE SCENE DISPLAY TECHNIQUE---The entire scene of 3D data can be displayed by the use of a vibrating mirror. A single slice of a 3D object can be displayed onto a 2D screen at a short time. If we then display many such slices in a rapid succession and view the screen through a vibrating mirror of variable focal length, we can make the slices appear to be in their correct spatial location relative to each other.

directional rotation has only one zero in the first row so the economy of calculations will be very small.

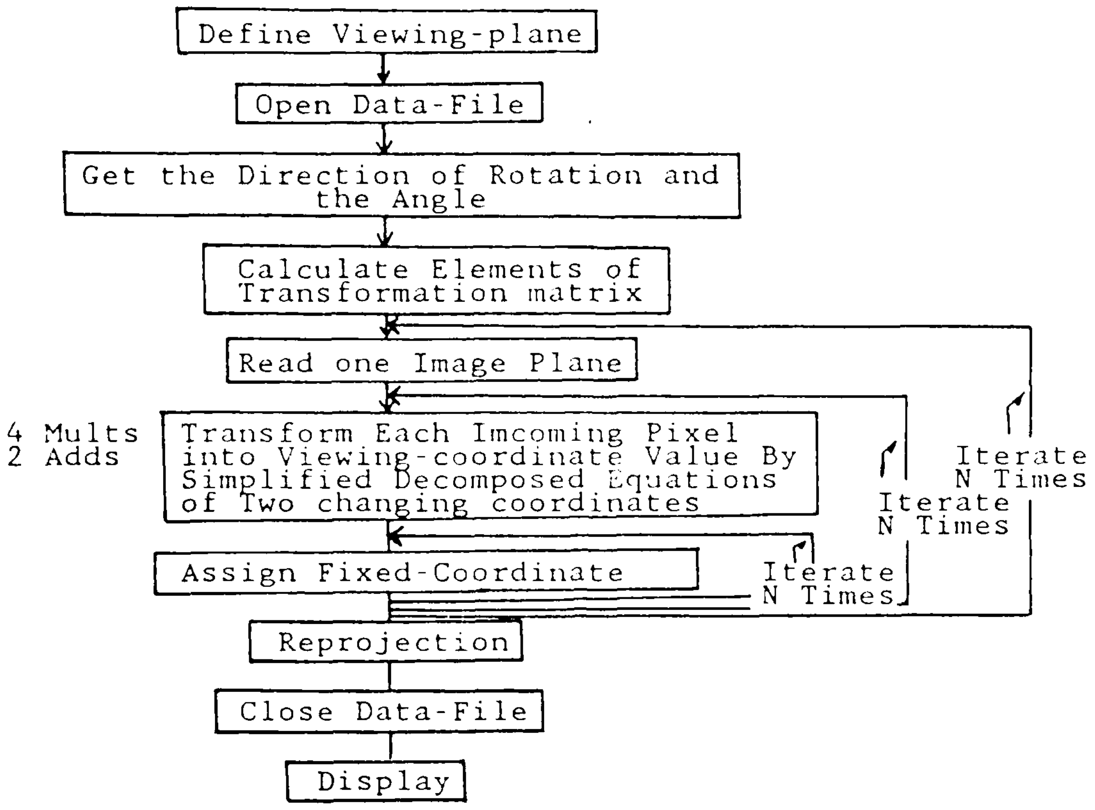

Define Viewing-plane
Open Data-File
Get the Direction of Rotation and the Angle
Calculate Elements of Transformation matrix
Read one Image Plane
4 Mults
2 Adds
Transform Each Imcoming Pixel into Viewing-coordinate Value By Simplified Decomposed Equations of Two changing coordinates
Assign Fixed-Coordinate
Iterate N Times
Iterate N Times
Iterate N Times
Reprojection
Close Data-File
Display

Figure 2.7 Flow-Chart of Decomposition with Fixed Coordinate(Rotation about One-axis).

3. Precalculation and Indexing Method

Another efficient method makes use of indexing instead of multiplication. This method is represented in the following equation

$$\begin{pmatrix} x\bullet \\ y\bullet \\ z\bullet \end{pmatrix} = \begin{pmatrix} c11 & c12 & c13 \\ c21 & c22 & c23 \\ c31 & c32 & c33 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} c11x + c12y + c13z \\ c21x + c22y + c23z \\ c31x + c32y + c33z \end{pmatrix} \qquad (2.3)$$

2. Decomposition with Fixed Coordinate

Let us consider a rotation about only one axis. As shown in Equation 2.1 the transformation matrix includes 4 zeros and a unit value in the main diagonal so the center axis of the rotation will not be changed

$$\begin{pmatrix} x\bullet \\ y\bullet \\ z\bullet \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x\cdot 1 & + & y\cdot 0 & + & z\cdot 0 \\ x\cdot 0 & + & y\cdot\cos\theta & - & z\cdot\sin\theta \\ x\cdot 0 & + & y\cdot\sin\theta & + & z\cdot\cos\theta \end{pmatrix} \quad (2.1)$$

$$\begin{pmatrix} x\bullet \\ y\bullet \\ z\bullet \end{pmatrix} = \begin{pmatrix} & x & \\ y\cos\theta & - & z\sin\theta \\ z\cos\theta & + & z\cos\theta \end{pmatrix} \quad (2.2)$$

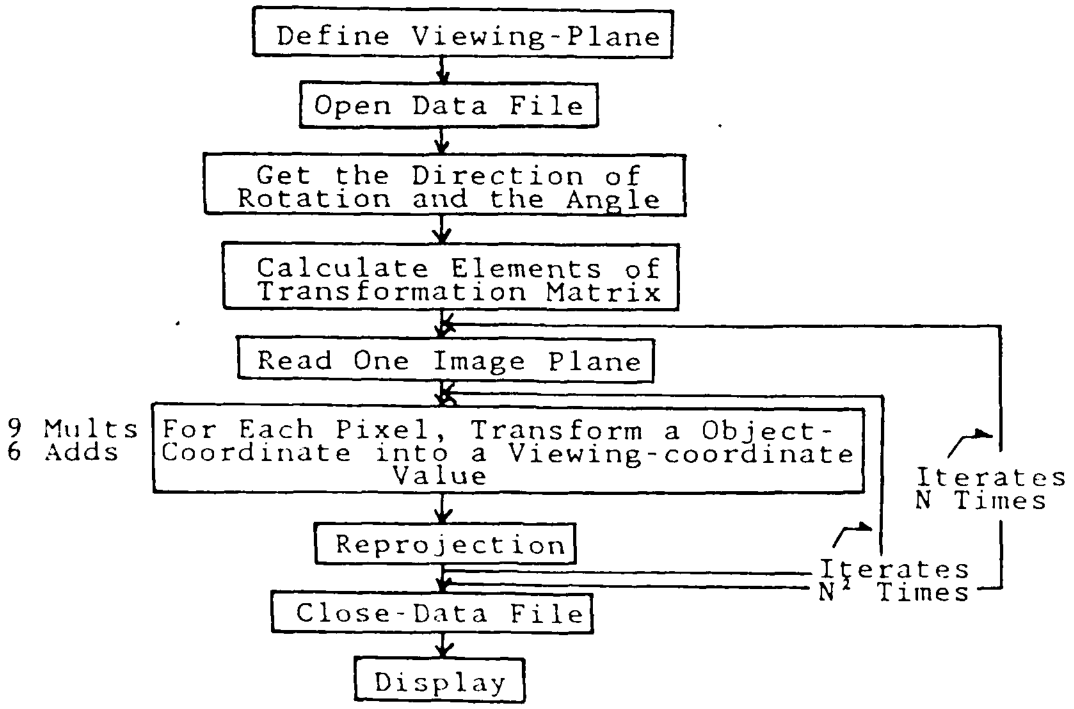
Equation 2.1 shows an example of rotation about the X-axis using the direct matrix multiplication method. Equation 2.2 suggests a more efficient implementation method. If the matrix multiplication is simplified to 3 equations as shown in Equation 2.2, it results in an economy of 4 multiplications and 4 additions for each pixel transformation.

Another point in comparing the two equations is that the first row of the transformation matrix consists of only a unit value and two zeros. This row maps a incoming vector into itself. (This is the fixed coordinate.) The other two coordinate values will be changed according to Equation 2.2. This method is represented in the flow chart shown in Figure 2.7. The inner loop does not include a call for a matrix multiplication operation, but the outer two loops call the subroutine 'Rotation' which includes matrix multiplication operations. Therefore the total number of calculations required are $4N^2$ multiplications and $2N^2$ additions.

This method of decomposition with fixed coordinates is not very helpful for the case of arbitrary directional rotation. The transformation matrix of the arbitrary

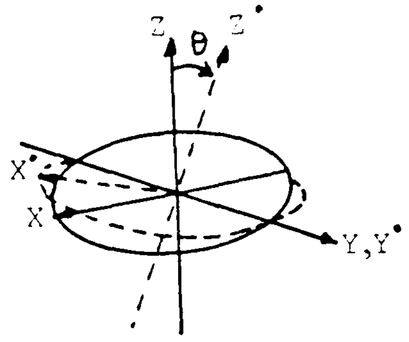
1. Direct Matrix Multiplication

Suppose we want to rotate a 3D object. Every data point in the object-coordinate has x, y, z coordinates which form a 3D position vector. To rotate this object we have to multiply the position vector by a 3x3 transformation matrix. As shown in Table 1, the transform of each vector requires 9 real multiplications and 6 real additions. If one dimension of the volume is N, the total calculation reaches $9N^3$ multiplications and $6N^3$ additions. The flow chart for implementing this method is shown in Figure 2.6. In a PASCAL environment, one record consists one plane and each record is accessed sequentially. Whenever an image plane is accessed the program rotates the coordinates of pixels contained in the plane one by one. Therefore the subroutine 'Rotation' is called N^3 times.



Define Viewing-Plane
Open Data File
Get the Direction of Rotation and the Angle
Calculate Elements of Transformation Matrix
Read One Image Plane
9 Mults 6 Adds
For Each Pixel, Transform a Object-Coordinate into a Viewing-coordinate Value
Iterates N Times
Reprojection
Iterates N² Times
Close-Data File
Display

Figure 2.6 Flowchart of the Direct Matrix Multiplication.



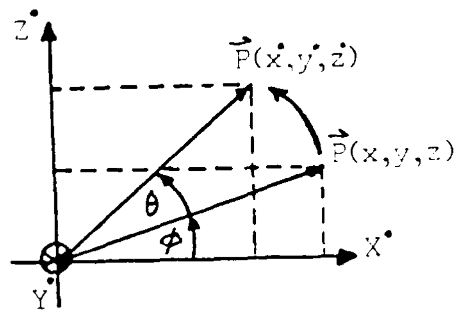


Figure 2.5 Rotation about Y-Axis.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \cos r & 0 & \sin r \\ 0 & 1 & 0 \\ -\sin r & 0 & \cos r \end{pmatrix} = \begin{pmatrix} \cos r & 0 & \sin r \\ \sin\theta\sin r & \cos\theta & -\sin\theta\cos r \\ -\cos\theta\sin r & \sin\theta & \cos\theta\cos r \end{pmatrix}$$

This matrix could not be applied in the reverse directional rotation, because a matrix multiplication AB is generally not equal to BA. Therefore we have to be careful in our selection of matrix multiplication sequences.

C. PROGRAMMING METHODS

General procedures for the image display are as follows: (1) Set up the object-coordinate value, (2) Calculate the corresponding viewing-coordinate value by coordinate transformation, (3) Access a pixel and place it in the corresponding viewing-coordinate, (4) Perform projection and display.

The following algorithms use different methods for coordinate transformation. Because coordinate transformation is the central issue in this problem the program efficiency will mainly depend on it.

$P(x,y,z) \Longrightarrow P^{\bullet}(x^{\bullet},y^{\bullet},z^{\bullet})$

$$\begin{pmatrix} x^{\bullet} \\ y^{\bullet} \\ z^{\bullet} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ y\cos\theta - z\sin\theta \\ y\sin\theta + z\cos\theta \end{pmatrix}$$

In figure 2.4, we let $x = x$, $y = \cos\phi$, $z = \sin\phi$. After rotation we have

$$\begin{aligned} x^{\bullet} &= x\ , \\ y^{\bullet} &= \cos(\theta + \phi) = \cos\theta\cos\phi - \sin\theta\sin\phi = y\cos\theta - z\sin\theta, \\ z^{\bullet} &= \sin(\theta + \phi) = \sin\theta\cos\phi + \cos\theta\sin\phi = y\sin\theta + z\cos\theta. \end{aligned}$$

This result is just the matrix multiplication, the product of a transformation matrix and a vector in the object-coordinate.

3. Rotation about Y-Axis

Rotation about the y-axis can be shown in the same manner as rotation about the X-axis above. (See Figure 2.5)

$P(x,y,z) \Longrightarrow P^{\bullet}(x^{\bullet},y^{\bullet},z^{\bullet})$

$$\begin{pmatrix} x^{\bullet} \\ y^{\bullet} \\ z^{\bullet} \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x\cos\theta + z\sin\theta \\ y \\ -x\sin\theta + z\cos\theta \end{pmatrix}$$

4. Rotation about Both-Axis

Rotation about an arbitrary axis requires a more complex operation. But we can restrict ourselves to two rotation directions for adequate display purposes. This combination of two directional rotations can be considered as a cascaded operation, first about the X-axis and then about the Y-axis or vice versa. If rotation about the X-axis, then rotation about Y-axis is chosen, then the rotation matrix will be

If the determinant of A is 1, it produces a pure rotation about the origin. Before considering rotation about an arbitrary axis, two special cases are examined: rotation about the X-axis (elevational) and rotation about the Y-axis(horizontal).

2. Rotation about X-Axis

In this case the rotation matrix will have zero in the first row and first column except a unit value on the main diagonal. The other terms are rotation dependent and will be determined by considering the specific rotation angle of a point. Figure 2.4 shows us this situation geometrically. Rotation about only one axis is the same as 2D transformation because the rotational axis is fixed in space. Here the rotation angle is defined to be positive in the counter-clockwise direction when a vector rotates.

* In this figure, $\vec{P}$ is an unit vector.

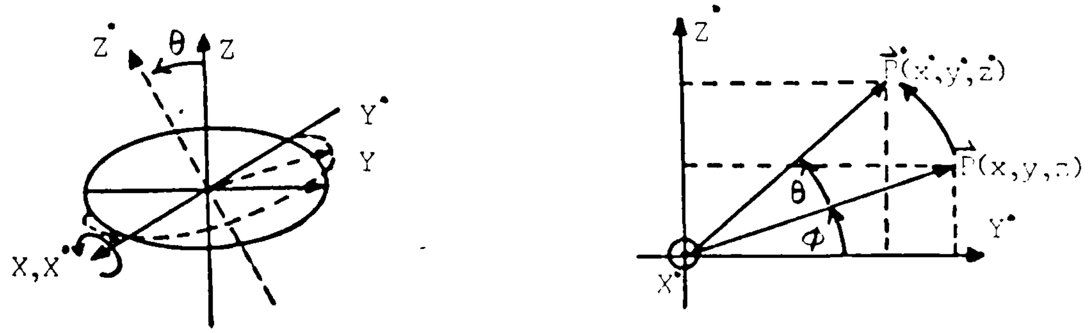


Figure 2.4 Rotation about X-Axis.

In the Fig 2.4, the X, Y, and Z represent the object-coordinates and X•, Y•, and Z• the viewing-coordinate system respectively. Thus our matrix transformation fomula is

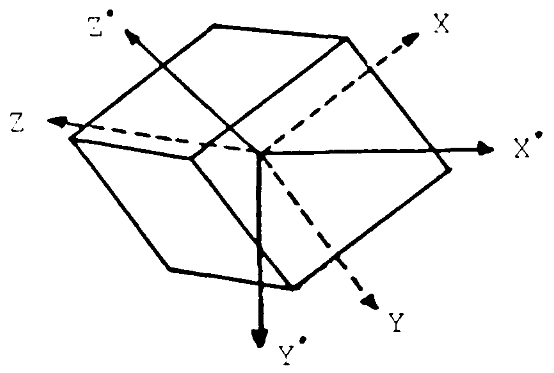


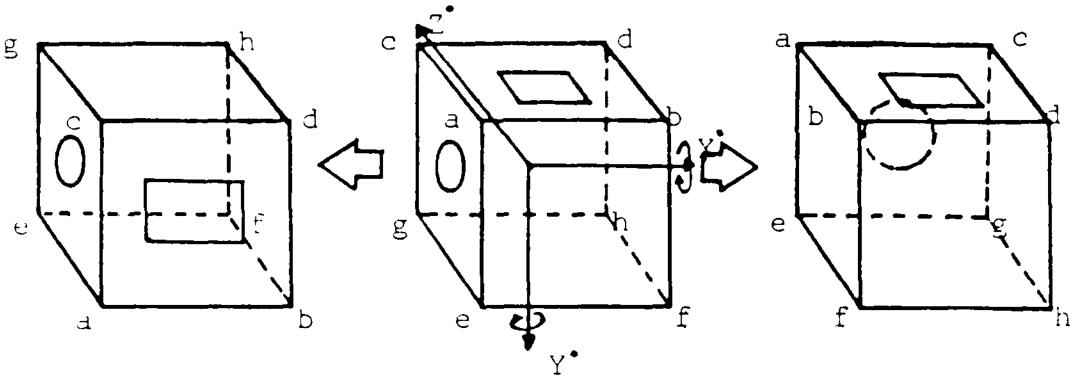
Figure 2.3 Coordinate System.

B. COORDINATE TRANSFORMATION

1. Matrix Multiplication

Usually a coordinate transformation is done by multiplying the original coordinate by a transformation matrix. If we use matrix notation this will result in a vector transformation from an R^3 space to another R^3 space. We can define a function T: $R^3 \dashrightarrow R^3$ by $T(X) = AX$ where A is a transformation matrix and X is a input vector. Observe that if X is a 3x1 matrix and A is 3x3 matrix, then the product AX is also a 3x1 matrix. Thus T maps R^3 into R^3. This kind of linear transformation is called 'matrix transformation'. Algebraically a 3D to 3D transformation matrix requires 3 basis functions. Suppose we want to rotate a 3D vector $X = x\hat{x} + y\hat{y} + z\hat{z}$. The transformation matrix for this operation should be a 3x3 matrix. In matrix notation we write

$$T\left(\begin{pmatrix} x \\ y \\ z \end{pmatrix}\right) = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{pmatrix}$$



(A)90 Rotation about X-Axis (B)Original Image (C)90 Rotation about Y-Axis

Figure 2.2 Direction of Rotation.

Rotation about both-axes is performed by the combination of elevational and horizontal rotations sequentially.

4. Coordinate System

Figure 2.3 shows the coordinate system we will use throughout this work.

A coordinate system is selected according to the right hand rule. As shown in Figure 2.3, the horizontal line represents the X-axis, the vertical line represents the Y-axis, and depth is labelled with the Z-axis. Object-coordinates are those we define relative to the object during image sampling, and viewing-coordinates are those defined for displaying the image. The viewing-coordinate system always corresponds to the coordinate of a display screen. Thus the orientation of the object with respect to the viewing-coordinates defines the rotation angle.

2. Center of Rotation

The center of rotation is arbitrary, although it is generally chosen to be near the center of the object (Figure 2.1). This is done so that after rotation the reconstructed image will not be shifted too much out of the effective viewing-window defined on the screen.

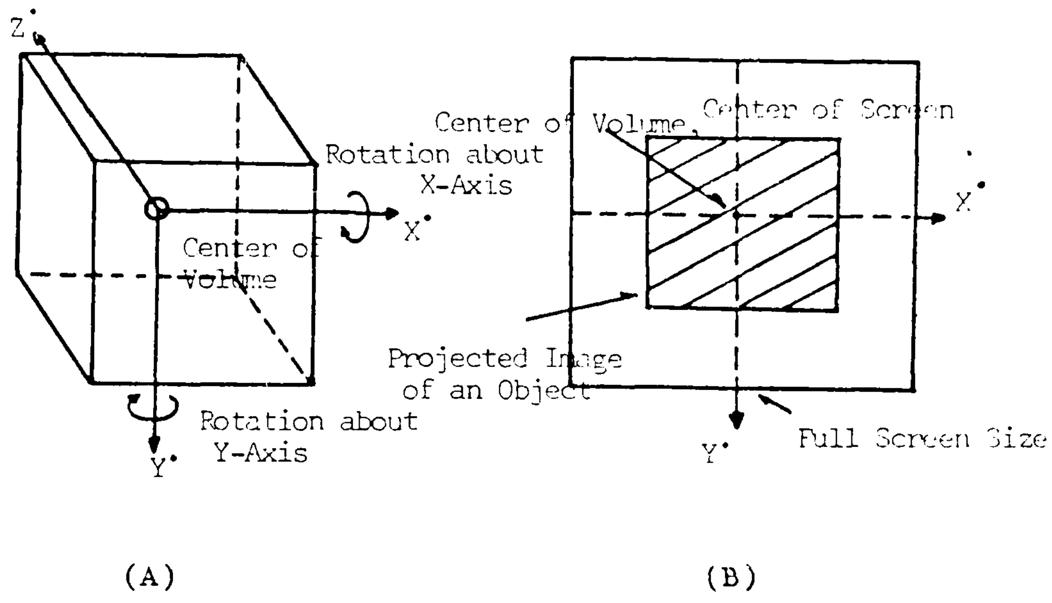


Figure 2.1 Center of Rotation(A) and Viewing-Plane(B).

3. Rotation Types

Three kinds of rotation are used for this work: Rotation about X-axis (Elevational), rotation about Y-axis (Horizontal), and rotation about both-axes(a combination of rotation about the X-axis followed by rotation about the Y-axis). These three kinds of rotation will be enough to provide a desired viewing-angle from any directions. Figure 2.2 illustrates examples of elevational and horizontal rotations. Figure 2.2(A) shows +90 degrees rotation about X-axis and Figure 2.2(B) shows +90 degrees rotation about Y-axis.

II. COORDINATE TRANSFORMATION

A. GENERAL

1. Introduction

Several methods have been proposed for the display of 3-dimensional(3D) information contained within a series of parallel Computed Tomography(CT) cross sections. This is referred to as a 3D or volume reconstruction problem. The reconstructed volume data will be displayed on a 2-dimensional(2D) TV screen using reprojection, dissection, and dissolution techniques. Although the problem of obscuration exists in the reprojected image, it is not as severe as in the case of conventional radiographs because the reconstructed volume can be viewed from different viewing-angles. Before reprojection the object can be mathematically oriented to a desired viewing-angle. The choice of desired viewing-angles will enable radiologists to discover the important information from the image.

The objective of this chapter is to present an example of 3D coordinate transformation. Because digital image data is comprised of a huge amount of 3D data points, our rotation algorithm must be efficient. Also, coordinate rotation invokes time-consuming floating-point operations in the computers. An inefficient algorithm will prove unwieldy and lead to an undesirably long processing delay before results are available. In the final part of this chapter, hardware alternatives to the software design which will implement the rotation algorithm developed in this chapter will be considered.

3. Numerical Dissolution

Selective numerical dissolution is a process whereby the relative contributions of selected voxels can be decreased before rotation. In this manner obscuring structures are only partially removed and this results in a 'see through' effect.

The criterion for numerical dissolution is based on the density difference between the structures within the reconstructed volume. The density threshold is identified empirically by first choosing an intitial threshold and then adjusting the value to achieve the desired dissolution effect. Once the desired threshold is identified, numerical dissolution is accomplished by dimming all voxels with values below threshold during reprojection. The dimming process is the same as that in the dissection process.

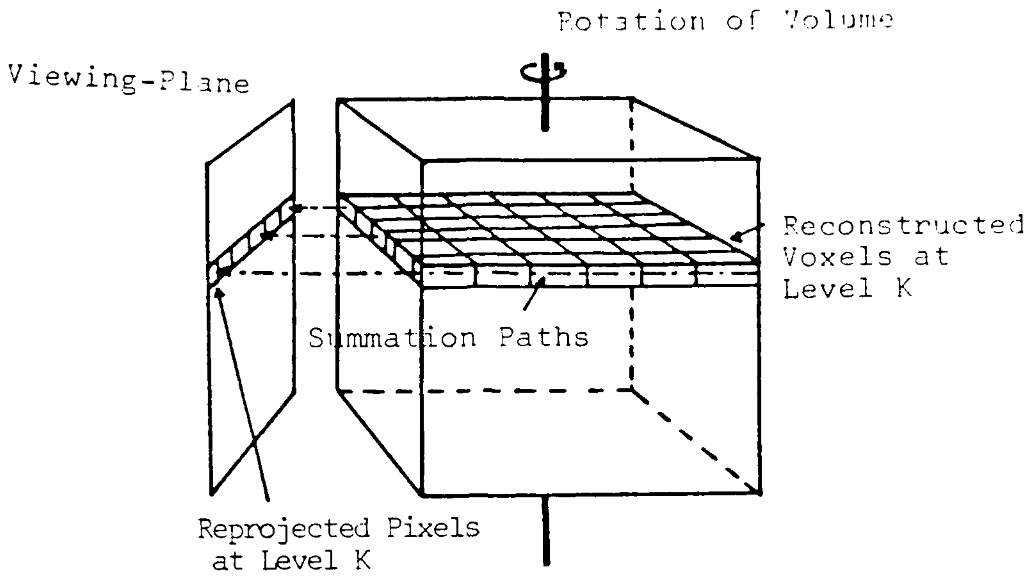

Rotation of Volume
Viewing-Plane
Reconstructed Voxels at Level K
Summation Paths
Reprojected Pixels at Level K

Figure 1.1 Reprojection Process.

The resulting 2D array of pixel values can be rescaled and displayed onto the specific display device. Subsequently it is viewed as an image on a TV monitor.

2. <u>Numerical Dissection</u>

The dissection technique is used to reduce the effect of obscuration and to clearly show an internal structure of the volume data. In this technique, a plane of dissection is highlighted by selectively dimming the voxels in front of the plane and ignoring all the voxels behind the plane. Dimming is accomplished by replacing the value of each voxel with the product of an original voxel density value and a constant less than unity. For example, a constant 0.1 would result in a intensity reduction to 10 % of its original value.

The result is that the internal structures at the plane of dissection are more clearly visible. It is also easy to see the spatial relationship of the object plane to those structures in front of the plane.

THE SCOPE OF THE STUDY---The study of techniques of orthographic reprojection of 3D objects after rotation to a selected viewing-angle is the focus in this work. The computer technique for single plane dissection is also incorporated here. Generally 3D volume data consists of a huge amount of voxels. Therefore, an efficient algorithm is essential for the manipulation of volume data.

After rotation to the selected viewing-angle, each voxel of the volume data does not lie on the grid of the viewing-coordinate system. In other words, each voxel looks like a polygon rather than a parallelepiped along the viewing-angle from the display screen. This will yield an obscuring effect between voxels during projection. Therefore, interpolation is necessary to resolve the contribution of voxels to each pixel in the viewing-plane.

The main emphasis in this work is on the study of efficient rotation algorithms and the interpolation method.

In Chapter II, various rotation algorithms are discussed, as well as the results obtained through the use of artifical test data.

In Chapter III different interpolation methods using various kernel functions are presented.

B. TECHNIQUES OF ORTHOGRAPHIC REPROJECTION

1. Reprojection Image Generation

The process of reprojection is illustrated in Figure1.1. A linear array of squares on the viewing-plane corresponds to the picture elements(pixels) at the level k of this cross-section. The intensity value of the pixels in the array is the sum of those voxels along the projection path which are perpendicular to the projection plane. When all volume elements at the level k have been projected then the voxels at level k+1 are projected.

Another way of displaying the whole scene is to project it onto the 2D screen. The work studied here is related to this approach.

SURFACE DISPLAY TECHNIQUE---Using the surface display technique, we are interesting in dipicting the appearance of surfaces of an object in the scene. This involves segmenting the original volume data into an object and a background, for example, by creating a binary image by which a 1 is assigned to a voxel in the object and a 0 to a voxel in the background. Various techniques such as 1D contour based, 2D surface based, and 3D parallelpiped based or sphere based display techniques are examples of surface displays.

REPROJECTION TECHNIQUE---There are two kind of techniques in the reprojection method: Orthographic reprojection and Radial reprojection. [Ref. 2]. Orthographic reprojection is performed numerically in the computer by summing the values of voxels along parallel paths through the reconstructed computed tomography(CT) scan image.

Radial reprojection is an alternate scheme in which the voxels of the CT scan reconstructed volume are projected onto a cylindrical surface surrounding the object, rather than orthographically onto a plane.

The reprojection of volume image data onto a 2D screen sometimes cannot provide sufficient information because the size of any given structures is not dependent on the distance from the observer. Perspective does not exist in the orthographic reprojection. Therefore it is necessary to rotate the volume image data to the desired viewing-angle before reprojection. In addition to the rotation, numerical dissection and numerical dissolution techniques are introduced to show selected portions of the reconstructed volume. [Ref. 2].

The elements(C_{ij}) of the transformation matrix are fixed after the rotation angle is specified, and the x, y, and z values of the object-coordinate iterate individually. We can calculate the multiplication of each coordinate and corresponding matrix element($C_{ij}X_K$) before accessing pixels of volume and then store those values in an array. After this, by indexing the array according to the incoming object-coordinate and adding these components together, we can calculate the viewing-coordinate value. Implementing this idea can allow us to avoid the time-consuming floating-point multiplications. It is represented in the flowchart of Figure 2.8. As shown in the figure the Indexing method does not use a 'Rotation' subroutine.

The array contents also can be calculated by successive additions as shown in the program 'Rotation_3_Dim'. When the multiplication of a coordinate and matrix-element is calculated, a coordinate value always increases by one unit. For example, $X_{K+1} := X_K + 1$, so the multiplication ($C_{ij}X_K$) becomes $C_{11}X_{K+1} = C_{11}X_K + C_{11}$, $C_{11}X_{K+2} = C_{11}X_{K+1} + C_{11}$ •••etc. Only the first term requires multiplication of 4 pairs of elements. The total number of additions required is $4N + 2N^2$. The 4N additions are the effective multiplications of the objective coordinate and transformation matrix element. The remaining $2N^2$ additions result from calculations of the viewing-coordinate value by means of double iteration loops.

Because all floating-point multiplications are avoided by indexing and adding, this method results in large processing time savings. The total number of calculations required is shown in Table 1. Compared to the other two methods, Table 1 indicates that this method is the most efficient of the three. But this method requires some working space in memory for storing the precalculated values. In a VAX-11/750 floating-point notation (where a floating point value occupies 4 bytes) 12N bytes are required for a rotation about one axis, and 24N bytes are required for rotation about both axes. This is a reasonable memory requirement considering the processing time savings realized by using this method.

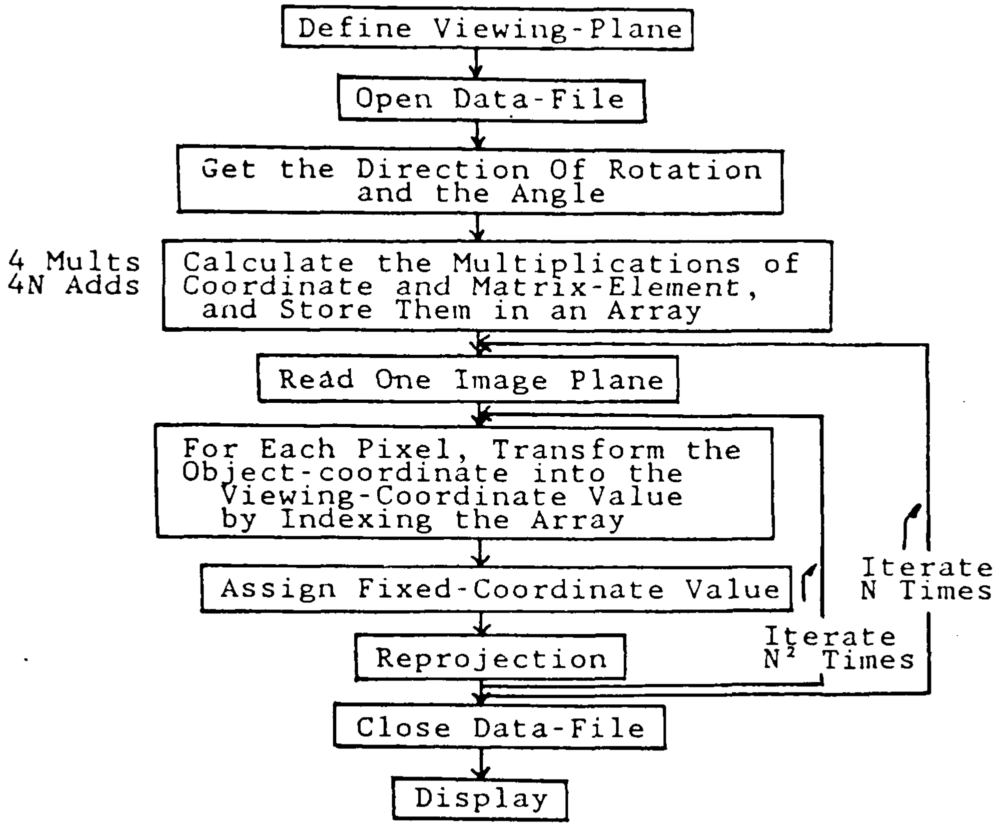

Define Viewing-Plane
Open Data-File
Get the Direction Of Rotation and the Angle
4 Mults 4N Adds
Calculate the Multiplications of Coordinate and Matrix-Element, and Store Them in an Array
Read One Image Plane
For Each Pixel, Transform the Object-coordinate into the Viewing-Coordinate Value by Indexing the Array
Assign Fixed-Coordinate Value
Iterate N Times
Reprojection
Iterate N² Times
Close Data-File
Display

Figure 2.8 Flow Chart of Precalculation and Indexing Method(Rotation about one-axis).

D. PROGRAM IMPLEMENTATION

The program 'Rot_3_Dim' which implements the Precalculation and Indexing method is included in Appendix A. In this PASCAL program, 'Objectcd' and 'Viewcd' represent the object-coordinate and the viewing-coordinate respectively. Data used to test the program is an artificially created double pyramid(N = 64). This 3D data is shown in Figure 2.9 which portrays the shape of a peak connected double pyramid. The density of the test data

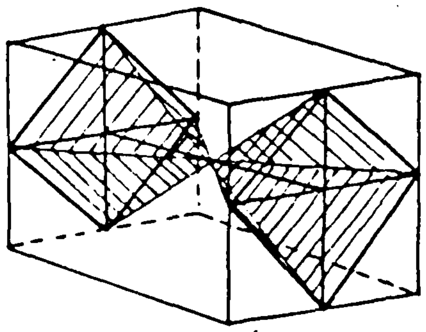
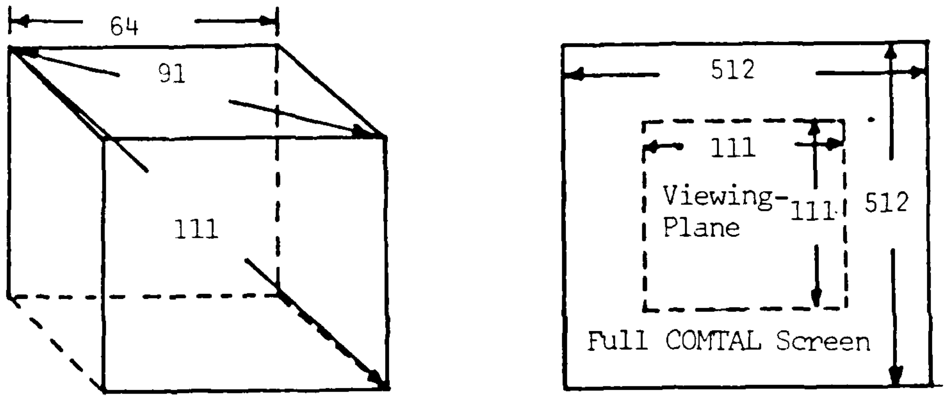


Figure 2.9 Double Pyramid.



(A) (B)

Figure 2.10 Test Data(A) and Display Buffer(B).

increases toward the center of pyramid. The object coordinate is defined so that the center of object is located at the center of the coordinate. The size along one dimension of the data is 64, but the program needs 111x111 elements size of a viewing-plane. In the worst case one side of the reconstructed image can have the diagonal size of the original image(Figure 2.10).

E. NUMBER OF CALCULATIONS AND PROCESSING TIME

The program 'Rot_3_Dim' which manipulates our test data is run on the DEC VAX-11/750 computer. Results are displayed on the COMTAL image processing system which itself has a PDP-11/23 processor. Table 1 shows a comparison of program efficiency and execution time among these 3 methods.

TABLE I

Table 1. Comparision of Methods

Direction / Methods	Rotation about One-Axis		Rotation about Both-Axis	
	NO of Cal.	Time Taken	NO of Cal.	Time Taken
Direct Matrix Multiplication	$9N^3$ Mults $6N^3$ Adds	over 25 secs	$9N^3$ Mults $6N^3$ Adds	over 10 mins
Decomposition with Fixed Coordinate	$4N^2$ Mults $2N^2$ Adds	25 Secs	$4N^2+4N^3$ Mults $2N^2+2N^3$ Adds	10 Mins
Precalculation and Indexing	4 Mults $4N+2N^2$ Adds (16N Bytes Storage)	15 Secs	8 Mults $9N^3$ Adds (32N Bytes Storages)	25 Secs

* N : The size of dimension.
All real value calculation.

F. HARDWARE REALIZATION

Modern computers have only one CPU. In this case each line of code must be executed sequentially. But digital image data is composed of a huge number of pixels. Therefore a special purpose computer having parallel processing capability is becoming popular. Here two simple hardware designs are suggested to realize the above algorithms of coordinate rotation. This hardware decreases the burden of iterative matrix multiplication in the coordinate rotation and releases the CPU to do more important task. Also it will increase the speed of the image rotation. These two design schemes use different components. Figure 2.11 illustrates the first hardware design scheme. The first design consists of 3 adders and 9 floating-point multipliers. An incoming X coordinate is multiplied with the first column of the transformation matrix, a Y coordinate the second column ,and a Z coordinate the third column respectively. Then the multiplied coordinates and matrix elements are added to make viewing-coordinate values as shown in Figure 2.11.

Figure 2.12 shows the second hardware design. The second one consists of 3 adders and 9 / 4N bytes memories instead of multipliers. Each set of the memory has the capacity of 4N bytes and individual memory element has 4 bytes width. The precalculated coordinates and matrix elements will be stored in the memories. These components are indexed according to the incoming object-coordinate values, and then added to calculate viewing-coordinate values. It is clear that an indexing is faster than a floating point multiplication. As may be expected there is a trade-off between cost and speed.

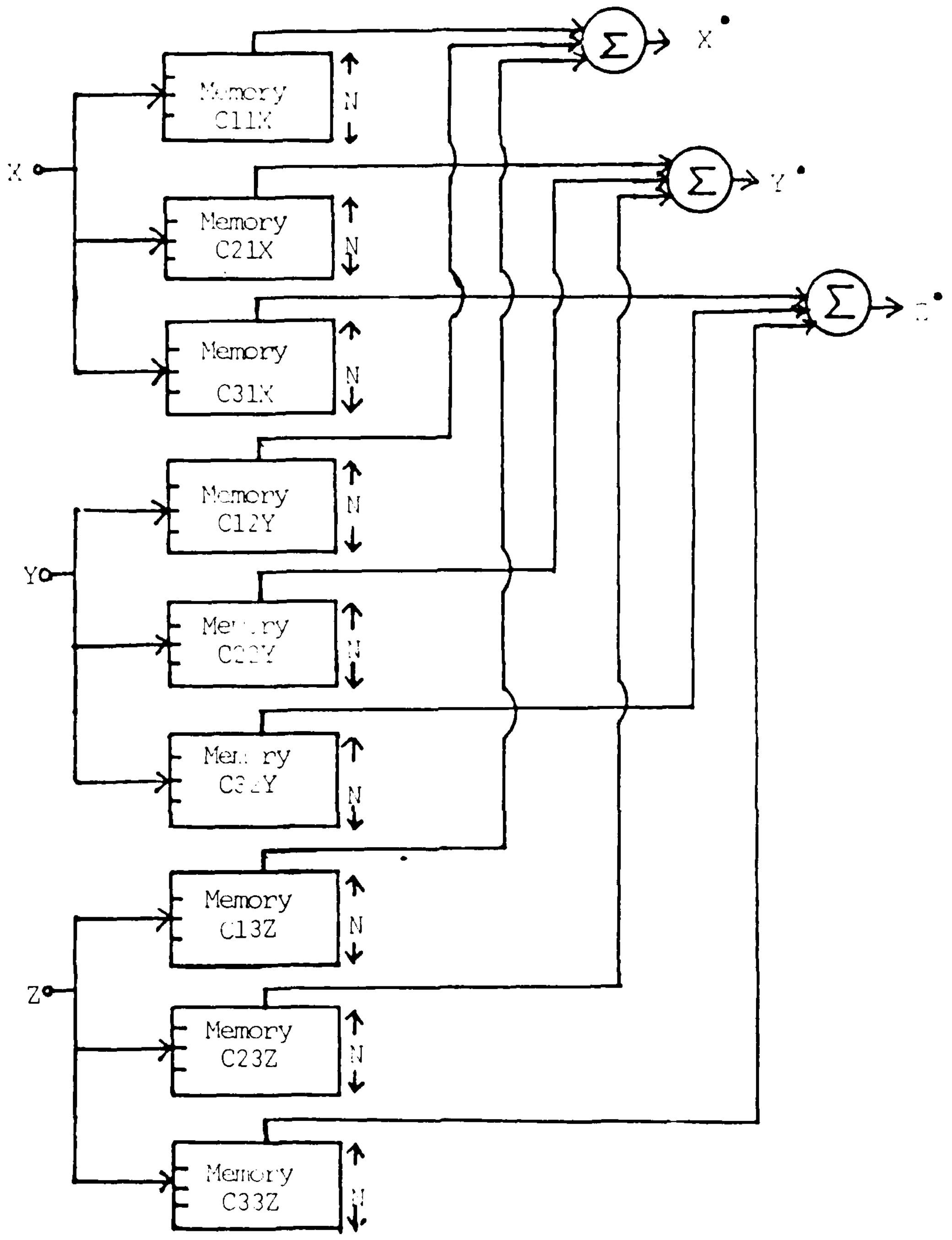


Figure 2.11 First schei e(9 / Floating-Point Multipliers).

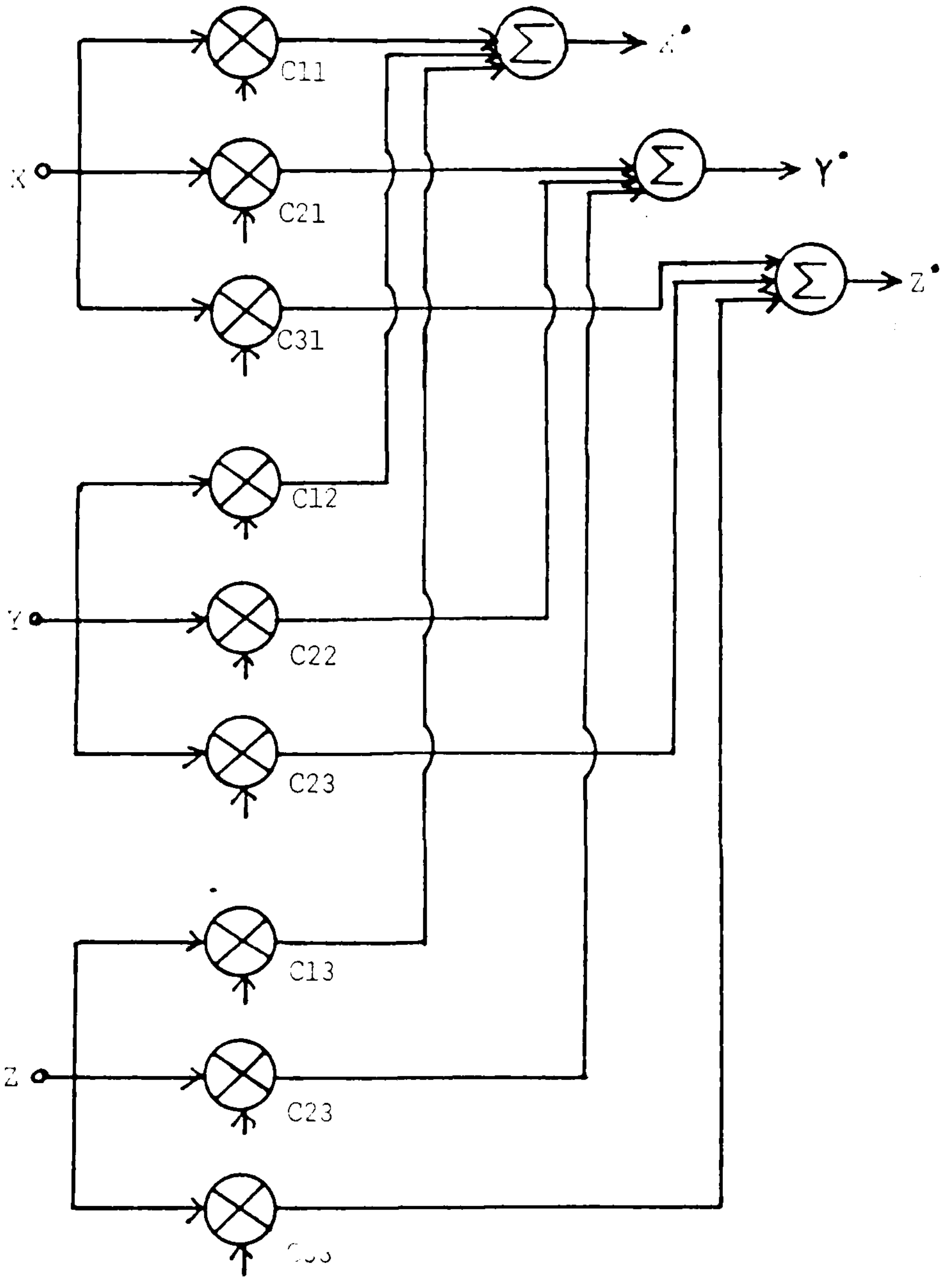


Figure 2.12   Second Scheme(9 / 4N Bytes Memories).

III. INTERPOLATION

A. INTERPOLATION IN IMAGE PROCESSING

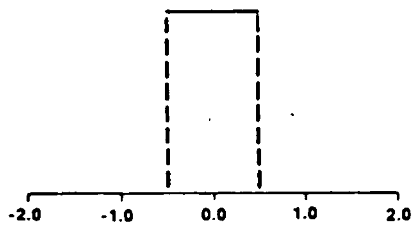
Interpolation is the process of estimating the intermediate values of a continuous signal from discrete samples. Interpolating is used extensively in digital image processing to magnify or reduce images and to correct a spatial distortion. In 3-dimensional(3D) digital image processing, each volume element(voxel) value represents the intensity value of a rectangular parallelepiped. This rectangular parallelepiped is not necessarily a cube, but it is assumed to be a cube that has equal length edges for our purposes here. A digital intensity value is generated by analog-to-digital conversion of a continuous intensity signal. Therefore, a digital data produced in this way can involve a certain amount of aliasing or blurring in the digitization process. But this effect is so small that we can ignore it. An exact image restoration will thus be possible if an appropriate interpolation method is applied. The goal of the interpolation of an encoded medical image is to reconstruct the original image as closely as possible. Because of the great amount of data associated with a digital image, an efficient interpolation algorithm is essential.

B. INTERPOLATION METHOD

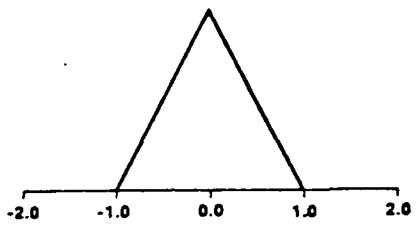
1. One-Dimensional Interpolation

An interpolation kernel function is a special type of approximating function. A fundamental property of an interpolation kernel function is that it must have the

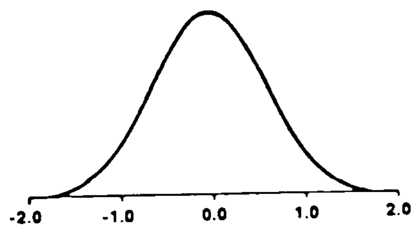
S(X) = SQUARE $\underbrace{******}_{\text{n times}}$ square, * : convolution



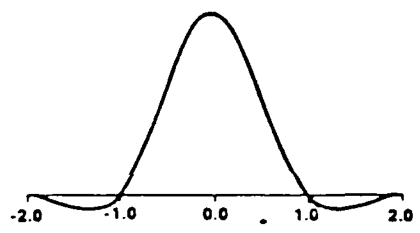
(A)Nearest Neighboring(n=1)



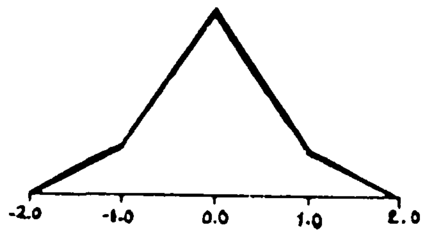
(B)Linear Interpolation(n=2)



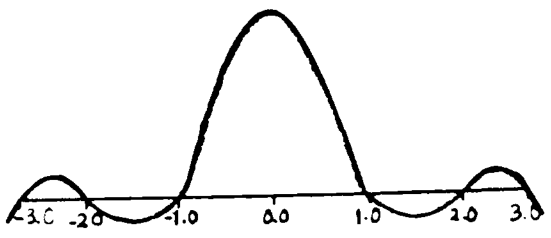
(C) Cubic Spline(n = 3)



(D) Cubic Convolution



(E) Piecewise Linear



(F) Sinc Function

Figure 3.1 Interpolation Functions(1D).

sampled data values at the interpolating knots or sampled points. In other words, if F is a sampled function and G is the corresponding interpolating function, then $G(X_K) = F(X_K)$ whenever X_K is an interpolation knot. For equally spaced data samples, many interpolation functions can be written in the form

$$G(x) = \sum_K C_K U\left(\frac{X - X_K}{h}\right) \qquad (3.1)$$

where h represents the sampling increment, the X_K's are the interpolation knots,and U is an interpolation kernel. The interpolation kernel in Equation 3.1 converts discrete data into a continuous function by an operation similiar to the convolution. Mathematically developed interpolation kernel functions include nearest neighboring, linear, cubic spline, cubic convolution, piecewise linear, and sinc function. A one-dimensional(1D) version of these interpolation functions are illustrated in Figure 3.1. [Ref. 3]. The first function in Figure 3.1 is a sample and hold, nearest neighboring, or replication interpolation. The second function is a square convolved with another square which results in a triangle or linear interpolation. The third function is the convolution of three squares or a cubic spline. The fourth function is an arbitrary piecewise linear function which illuminates the arbitrariness of the interpolation kernel. The fifth function is a cubic convolution kernel which is composed of piecewise cubic polynomials defined on the subintervals (-2,-1), (-1,0), (0,1), and (1,2). [Ref. 3]. The last one is the sinc function which provides an exact reconstruction. The use of the sinc function (sinX/X) as an interpolant, which has negative valued side lobes, requires an infinite number of terms. Realization is difficult. difficult in the realization. Polynomials of order two or greater can also be employed as an interpolation kernel function. The amplitude spectra of the nearest-

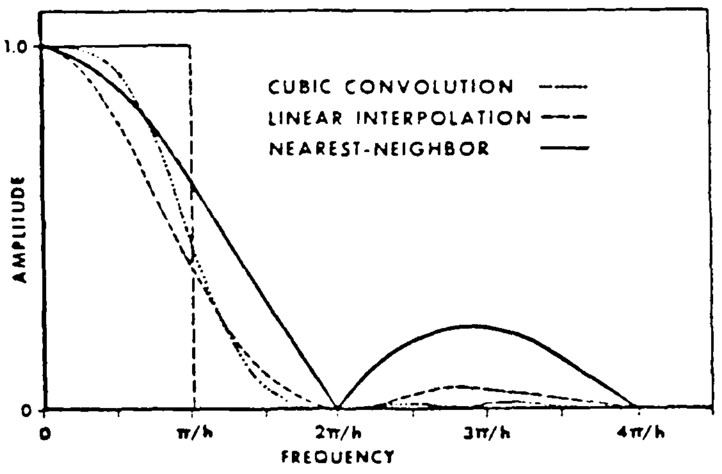
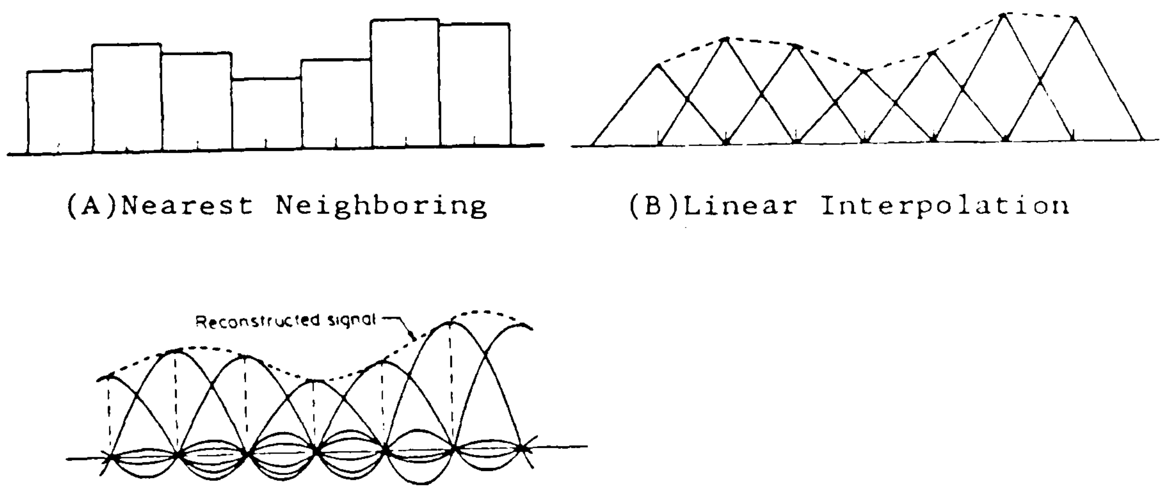


Figure 3.2 Amplitude Spectra of Interpolation Functions.



(A)Nearest Neighboring (B)Linear Interpolation

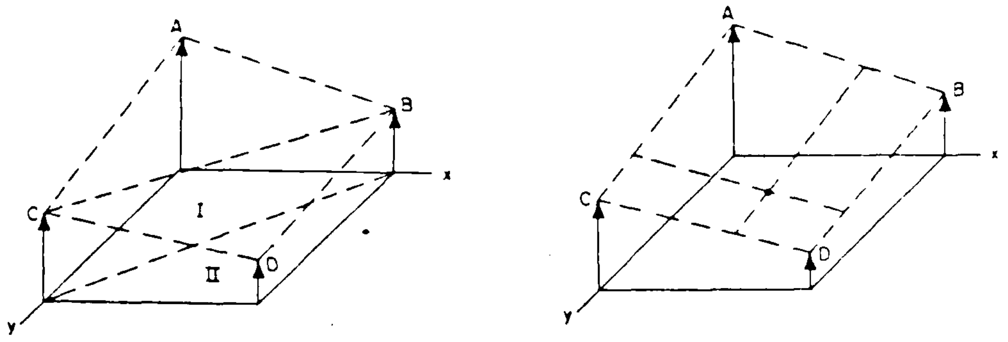
(C)Sinc Function

Figure 3.3 One-Dimensional Interpolation Process.

neighboring, linear interpolation, and cubic convolution interpolation kernels are shown in Figure 3.2 for frequency from 0 to 4π/h (where h is the sampling interval).

[Ref. 3]. The response of an ideal interpolation kernel is a unit step function in the frequency domain. In image data, the loss of high frequency information causes the image to appear blurred. On the other hand, deviation from the ideal spectrum beyond the shaded area contributes to aliasing. One-dimensional interpolation examples with several interpolation kernels are performed in a fashions shown in Figure 3.3. [Ref. 4]. Interpolation kernels have a significant impact on the numerical behavior of the interpolated functions. Because of their influence on accuracy and efficiency, it is necessary to select carefully an appropriate interpolating kernel for an image processing.

2. Two-Dimensional Interpolation



(A)Piecewise Linear Interpolation (B)Bilinear Interpolation

Figure 3.4 Two-Dimensional Linear Interpolation.

A two-dimensional(2D) interpolation is accomplished by two seperate 1D interpolations with respect to each coordinate. It should be performed along separable orthogonal coordinates of the continuous signal. A 2D linear interpolation kernel function is an example of an orthogonally separable interpolation function: $G(x,y) = G(x)*G(y)$. Figure 3.4 shows two examples of the linear interpolation method in 2D situations. [Ref. 4]. Example (A) shown in Figure 3.4 is performed in a piecewise fashion. In region I of example (A), points are linearly interpolated in the plane defined by pixels A,B,C, while in region II, the interpolation is performed in the plane defined by pixels B,C,D. The continuous bilinear interpolation is shown in example (B). It is done by linearly interpolating points along separable orthogonal coordinates of the continuous image signal.

C. PROBLEMS IN 3-DIMENSIONAL DATA PROJECTION

A 3D data is a collection of discrete values resulting from equalspace sampling. This data consists of small identical parallelepipeds (voxels) divided by three sets of planes parallel to the X, Y, and z axes. Each voxel has a sample value. It is referred to as the intensity value of a voxel. For clarity of discussion the voxel value is located on the grid of the object-coordinate system. Figure 3.5(A) shows the relationship between the voxels and the object-coordinate system before rotation. Orthogonal projection along the Z-axis is performed by integrating the intensity value of voxels along a line parallel to the Z-axis(any point P lying in one of the X-Y plane in the object-coordinate can not be obscured with another point on the same plane). The object-coordinate grids exactly correspond to the viewing-coordinate grids.

After rotation, the object-coordinate grid does not share the same location as the viewing-coordinate grid as shown in Figure 3.5(B). The coordinate value of a voxel on the grid of the object-coordinate will be transformed to its viewing-coordinate values. These values may not be integers which are the viewing-coordinate grids. This is the reason why we have to interpolate intermediate intensity values at the viewing-coordinate grids.

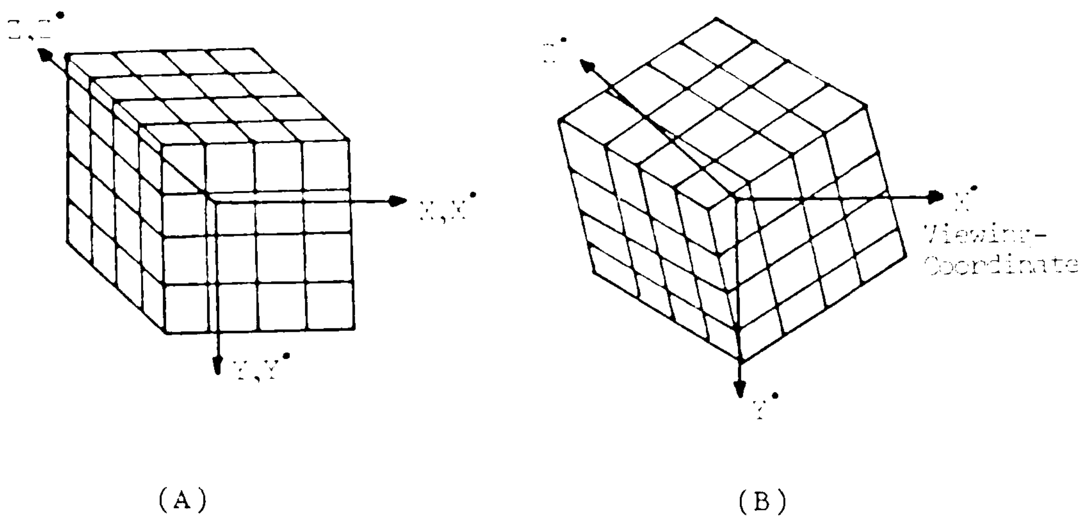


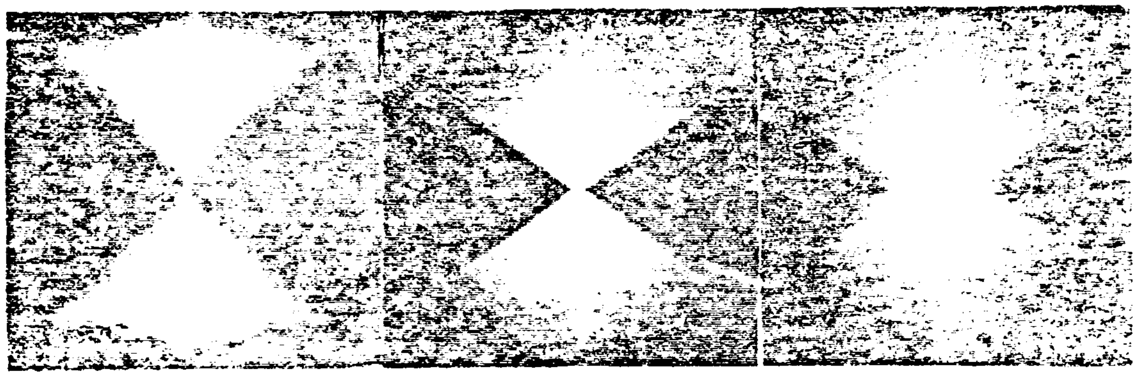
Figure 3.5 Relationship between 3-Dimensional Data and Viewing-Coordinate Before Rotation(A) and After Rotation(B).

To reconstruct the exact object, a high order interpolation kernel such as sinc functions will produce better results. But using a high order kernel requires many calculations and sometimes it exceeds reasonable computer capability. If we can achieve an appropriate result with a low order kernel function, We can realize a trade-off between fidelity and processing time. For that reason, only the linear interpolation kernel in the 2D and 3D cases will be used in this work.

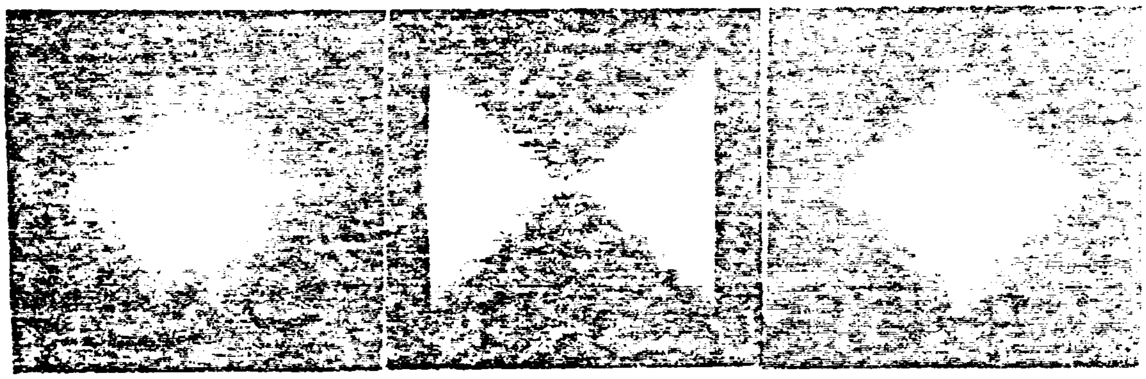
D. APPLICATION OF LINEAR INTERPOLATION

1. <u>One-Dimensional Interpolation</u>

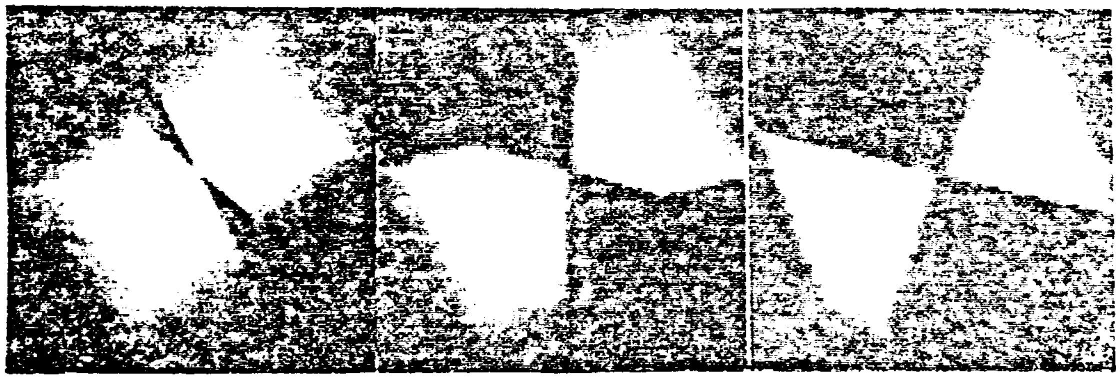
A linear interpolation function will yield lower quality results compared to that of the high order interpolation function . An intermediate interpolated point is calculated with the nearest two sample points. Linear interpolation includes an assumption that the two adjacent sample points have a linear relationship.



30 Degrees 45 Degrees 75 Degrees
(A) Rotation about X-Axis



15 Degrees 90 Degrees 180 Degrees
(B) Rotation about Y-Axis



30-30 Degrees 45-45 Degrees 60-60 Degrees
(C) Rotation about Both-Axis

Figure 4.3 Displayed Images by Reprojection.

The resulting pixel values in the viewing-plane will exceed the allowed pixel values of the display device so that the pixel should be rescaled to fit the specific display device requirement.

Several reprojected images with sphere-shape kernel are illustrated in the Figure 4.3. Different viewing-angles provide more informations about the structures of 3-dimensional data.

Before rotation, every voxel is arranged along viewing-coordinate so object-coordinate grids correspond to viewing-coordinate grids.

The reprojection can be performed as in the following procedure:

(1) For every incoming voxel, read the viewing-coordinate $X_K{\bullet}$, $Y_K{\bullet}$, then sum the density value onto a pixel corresponding at $X_K{\bullet}, Y_K{\bullet}$ in the viewing-plane (array: 'Viewpln').

Viewpln($X_K{\bullet}, Y_K{\bullet}$) := Viewpln($X_K{\bullet}, Y_K{\bullet}$) + Pixel($X_K{\bullet}, Y_K{\bullet}, Z_K{\bullet}$);

(2) Continue to reach the last voxel.

After rotation, it is not clear what portion of a voxel will contribute the pixel value in the viewing-plane because each voxel looks like a polygon to the sight of view. Through the interpolation method discussed in the Chapter III, the appropriate portions of the voxel value are added to the pixel values in the viewing-plane (see subroutine 'Projection' and 'Dissection' of the program 'Rot_3_Dim').

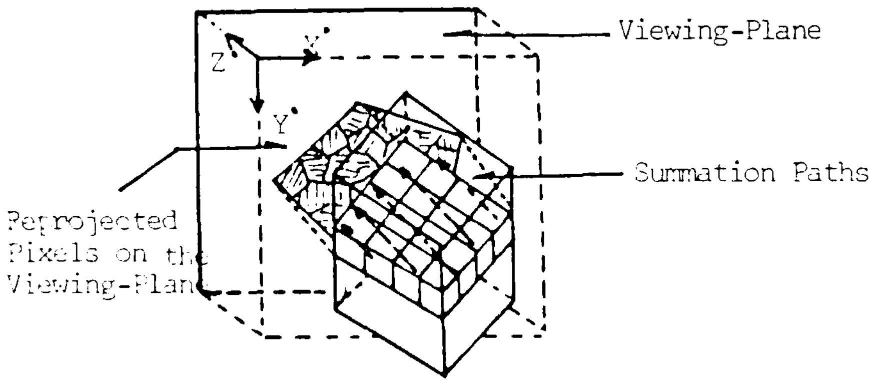


Figure 4.2 Reprojection Process after Rotation.

IV. PROGRAM IMPLEMENTATION OF REPROJECTION AND DISSECTION

A. ORTHOGRAPHIC REPROJECTION

Orthographic reprojection is performed numerically in the computer by summing the value of voxels along parallel paths through the reconstructed volume. The reprojected image constitutes a 2-dimensional image on the screen.

Figure4-1 illustrates the orthographic reprojection process onto the X•-Y• plane along Z• coordinate.

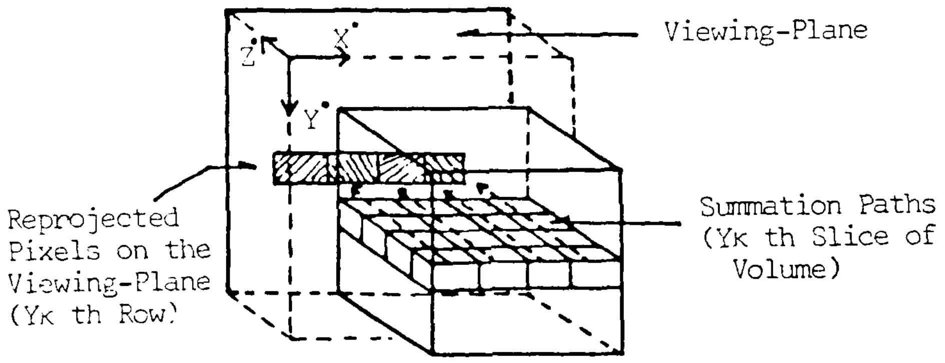

Viewing-Plane
Z•
X•
Y•
Reprojected Pixels on the Viewing-Plane (Yκ th Row)
Summation Paths (Yκ th Slice of Volume)

Figure 4.1 Reprojection Process before Rotation.

To implement this process in the program, first a viewing-plane (2-dimensional array: 'Viewpln') is defined parallel to the viewing-coordinate X•-Y• shown as in program 'Rot_3_Dim'. The viewing-plane composes of X•(column) and Y•(row) coordinates and lies orthogonally to Z•-coordinate (X•,Y•,Z• are viewing-coordinates). The Yκ'th slice of the volume is projected onto the Yκ 'th row of pixels in the viewing-plane as shown in Figure 4.1.

3. Three-Dimensional Interpolation

As dicussed in the 2D situation, linear interpolation using a cone-shape kernel is the most appropriate method. Because the shape of this kernel is invariant with respect to the direction, it can be easily extended to the 3D interpolation. Applying the linear interpolation method to the 3D situation requires a very complicate procedure. After rotation, a parallelepiped looks like a polygon instead of a cube to the sight of the viewer. Therefore, performing linear interpolation to the sphere is easier than to the polygon. In this case, only grids of object-coordinate inside of the sphere are affected by a pixel value lying at the center of a sphere. The algorithm for this method is :

(1) Consider one object-coordinate grid.

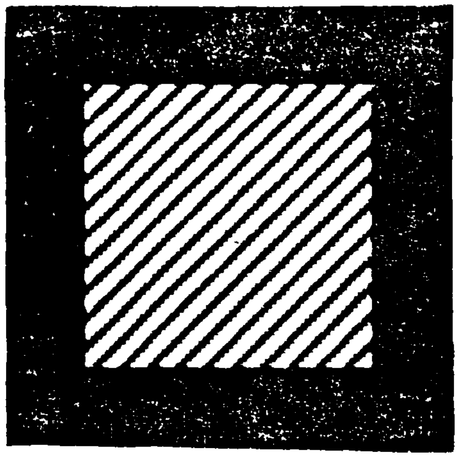
(2) Transform it into viewing-coordinate value by coordinate rotation.

(3) Calculate distance from it to the surrounding 8 object-coordinate grids.

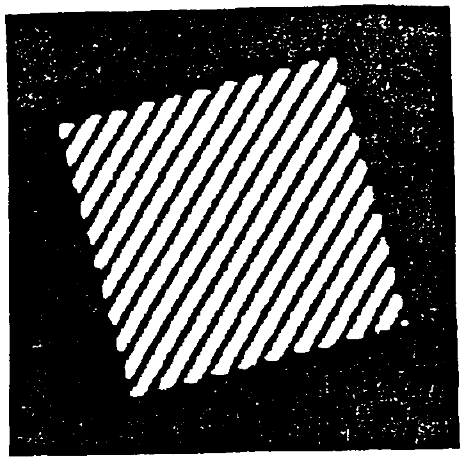
(4) Calculate the distribution of intensity value at the 8 object-coordinate grids when a distance is smaller than unity. Otherwise ignore it.

(5) Sum the distribution components of the 8 surrounding intensity values to calculate the value at a object-coordinate grid.

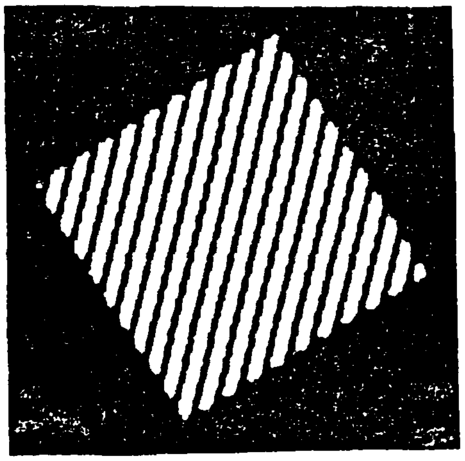
The disadvantages of this method are (1) Necessity for many floating-point multiplications, (2) Total error is the sum of inherent linear interpolation error and sphere-shape kernel error. More details will be discussed in the Chapter IV.



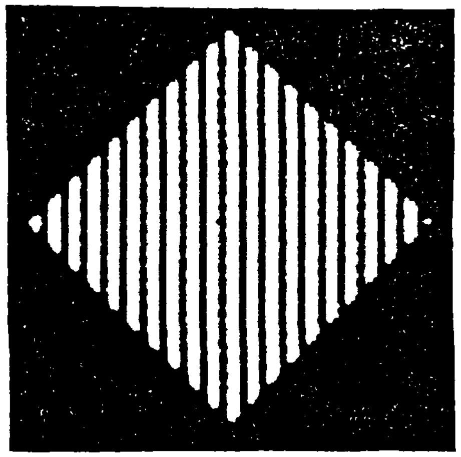
(A) Original Image



(B) 15 Degrees Rotation



C) 30 Degrees Rotation



(D) 45 Degrees Rotation

Figure 3.13 Displayed Images by Interpolation Using Cone-Shape Kernel.

The disadvantages of this method are: (1) It involves many multiplications to calculate the distances. (2) Error includes inherent linear interpolation error plus the cone-shape kernel error. The advantage of this method is that it can easily be applied in a 3-dimensional situation.

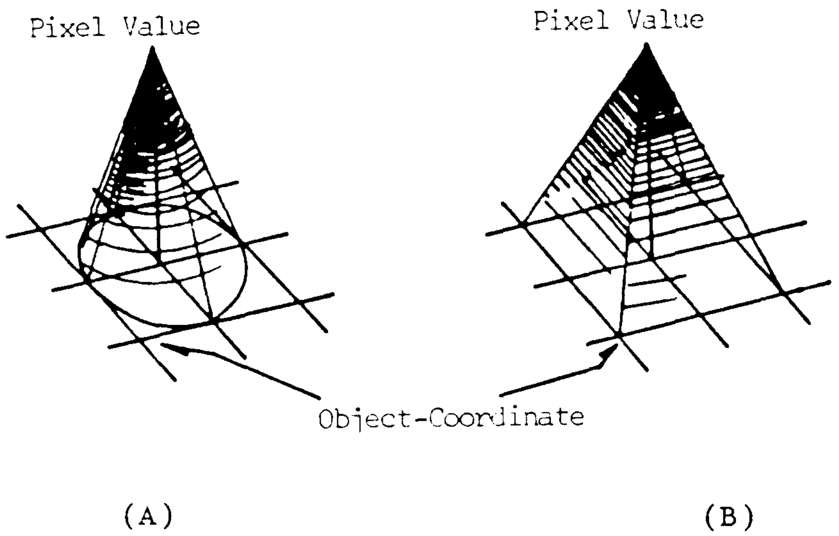


Figure 3.12 (A) Cone-Shape Kernel (B) Rectangular kernel.

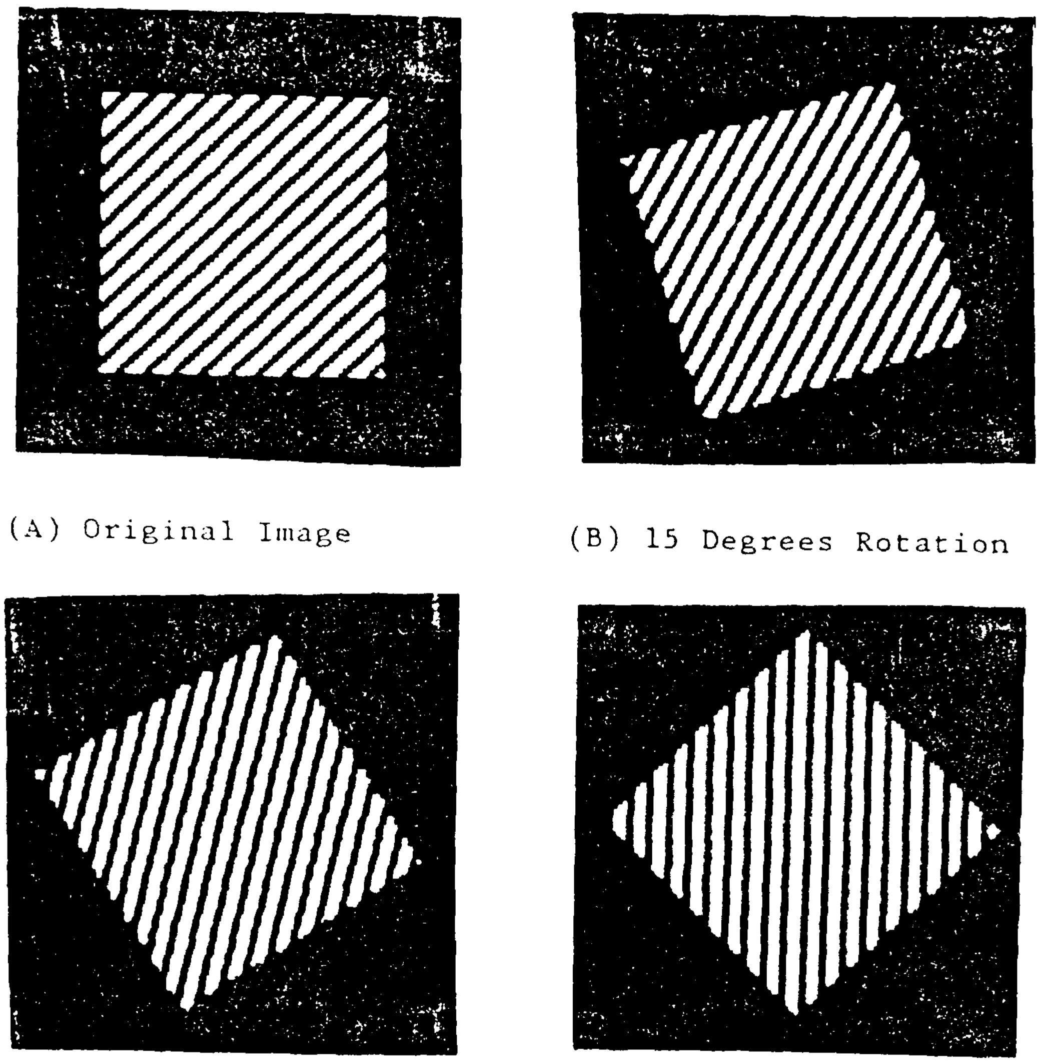
(2) Transform it to the viewing-coordinate value by coordinate rotation.

(3) Calculate the distance from it to the 4 surrounding object-coordinate grids.

(4) If a distance is smaller than unity, calculate the distribution component of a intensity value inversely proportional to the distance. Otherwise assign zero.

(5) Sum the distributions of all points to get the intensity value at the grid of the object coordinate.

Using this kernel, only viewing-coordinate grids inside the kernel are affected by the sample value at the center of the cone. Otherwise it is ignored. Figure 3.13 indicates that the error introduced by using the cone-shape kernel is not severe. Resulting displayed image is almost the same as the above one. Because this cone-shape kernel is invariant to the direction of rotation, it can be extended easily into a sphere-shape kernel in a 3D situation.



(A) Original Image (B) 15 Degrees Rotation

(C) 30 Degrees Rotation (D) 45 Degrees Rotation

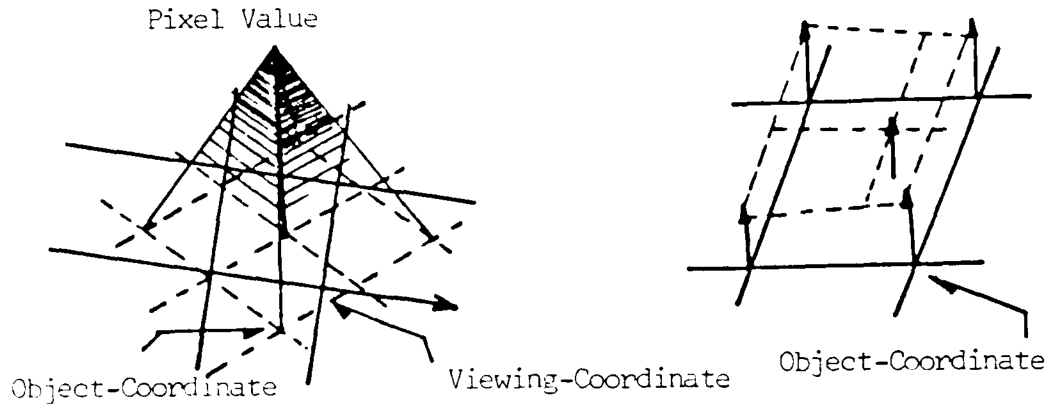
Figure 3.11 Displayed Image by Interpolation of the Signal from the Nearest 4 Samples at the Grids of the Object-Coordinate.

(1) It is invariant to the direction of rotation.

(2) Interpolation is performed on the object-coordinate

The algorithm for this method is

(1) Consider one object coordinate grid.



(A) (B)

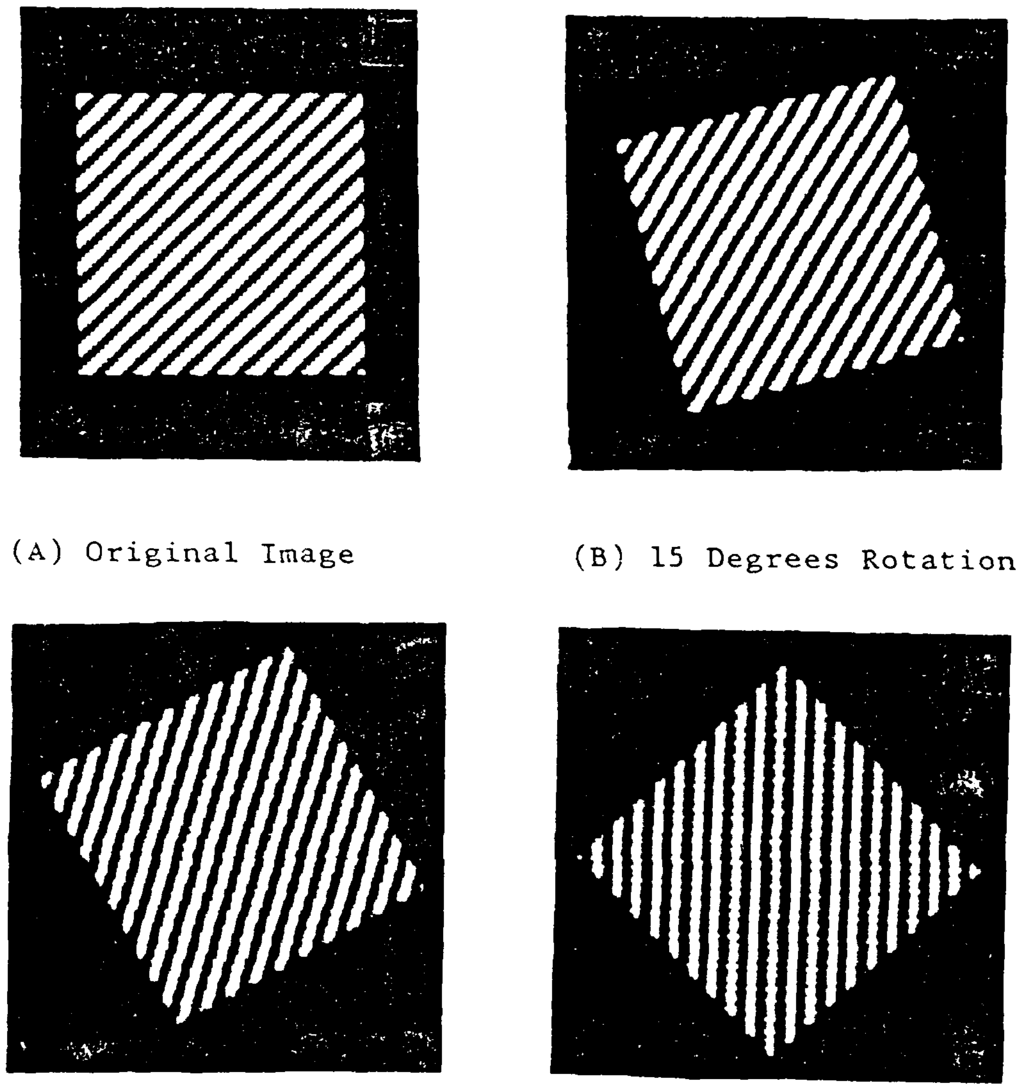
Figure 3.10 Interpolation Kernel over object-coordinate(A) and Bilinear Interpolation(B).

values. Therefore this method is not practical for manipulation of 3D data.

A displayed image obtained by this method is shown in Figure 3.11. It shows a better result than the interpolation after rotation of object-coordinate. Blurring effect is due to inherent linear interpolation error.

c. Interpolation Using Cone-Shape Kernel

We have already disscussed two interpolation methods. But these two methods have flaws which make it difficult to apply it to the 3D situations. The third method which avoids these flaws uses a 2D cone-shape kernel. Figure 3.12 illustrates how the cone-shape kernel is used for 2D linear interpolation. The density distribution of a pixel in the exact 2D linear interpolation should be done like Figure 3.12(B). But the cone-shape density distribution method is used for this purpose. The advantage of using a cone-shape kernel is



(A) Original Image (B) 15 Degrees Rotation

(C) 30 Degrees Rotation (D) 45 Degrees Rotation

Figure 3.9 Displayed Image with the Interpolation by Distributing the Sample Values to the Grids of the Viewing-Coordinate.

simultaneously in computer memory, which requires large working space.

Even with a large computer system, individual working space is not usually large enough for storing all data pl• = [Po * (l-(vx-ix))][l-(vy-iy)]

P2• = [Po * (l-(vx-ix))][vy-iy]

P3• = [Po * (vx-ix)][(vy-iy)]

P4• = [Po * (vx-ix)][l-(vy-iy)]

Where Pl•,P2• ,P3• ,P4• are distributed components of the 4 neighbors of the original sample Po located at (vx,vy). The displayed image thus obtained is shown in the Figure 3.9. This figure indicates that the error is proportional to the rotation angle and reaches a maximum at 45 degrees rotation. As shown in the lD interpolation case, the pixel value of the rotated object-coordinate grid will be projected orthogonally onto the viewing-plane (x•-y• plane). Even though the object coordinate grids are evenly spaced, the projected grids become compressed on the viewing-plane. But this method assumes the evenly spaced grids regardless of a rotation angle. Since the amount of error introduced depends on the rotation angle, it is impractical to apply it in the real display.

b. Interpolation of the Signal from the Nearest 4 Samples at the Grids of the Object-Coordinate

In this algorithm, interpolation is performed in bilinear fashion on the object-coordinate. The algorithm for this method is

(1) Consider one viewing-coordinate grid.

(2) Transform it into object-coordinate value by an inverse coordinate rotation.

(3) Perform the interpolation from its neares. 4 surrounding samples by the bilinear method.

This is a correct linear interpolation. However problems with this algorithm are (1) We have to rotate all viewing-coordinate grids. (2) All input data should be accessed

a. Interpolation by Distributing the Sample Values to the Grids of the Viewing-Coordinate

Since the data values are samples on the grids of the object-coordinate, interpolation of the intermediate signal should be done in the object-coordinate. But for the practical reason of data storage the interpolation by density distribution in the viewing-coordinate is attempted first. In this case, a linear interpolation kernel on the viewing-coordinate is assumed similiar to the kernel on the object-coordinate. The algorithm for this method is

(1) Consider one object-coordinate grid.

(2) Transform it into a viewing-coordinate value by coordinate rotation.

(3) Distribute a pixel value inversely proportional to the distance from the nearest 4 surrounding viewing-coordinate grids. (see Figure 3.8(a))

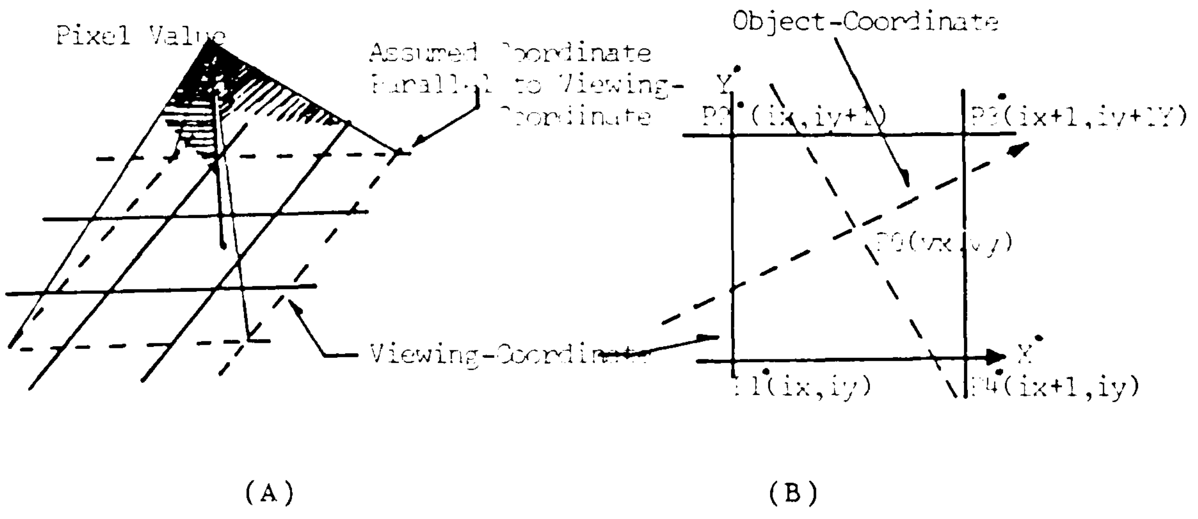


Figure 3.8 Interpolation Kernel over Viewing-Coordinate(A) Sampling Value Distribution(B).

As shown in Equation 3.2 and 3.3, each interpolation requires 2 multiplications, 1 addition, and 2 subtractions.

2. Two-Dimensional Interpolation

The two-dimensional interpolation is an extension of a 1D interpolation because the signal function is defined in the two orthogonal coordinates as $G(x,y) = G(x)G(y)$. Here three algorithms are tested with a sinusoidal 2D function having 64^2 elements: $F(x,y) = 127\cos(x+y) + 127$. For this example an identical sampling interval in both X, and y coordinates are used. The center of rotation is always taken to be the center of the coordinate systems. The rotation is applied to the function which is written in object-coordinate(X,Y). After rotation the sample points of the original function no longer coincide with the grid of viewing-coordinates. It is necessary to interpolate the signal value at the grids of the viewing-coordinate.

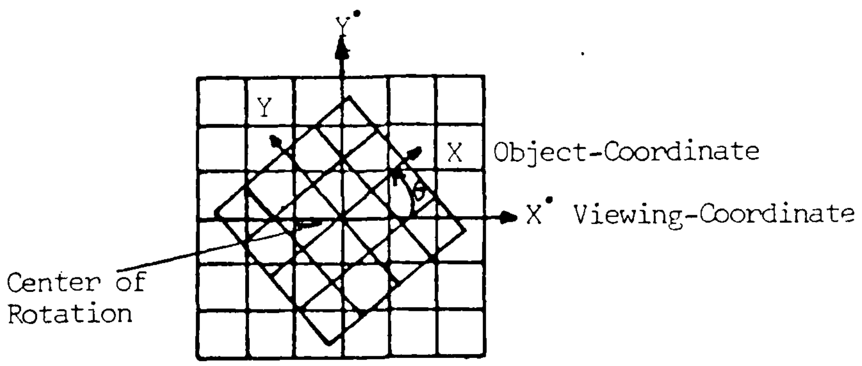


Figure 3.7 Two-Dimensional Rotation.

Let us consider a linear interpolation example of a signal after rotation in a 1D case. Figure 3.6 illuminates this situation.

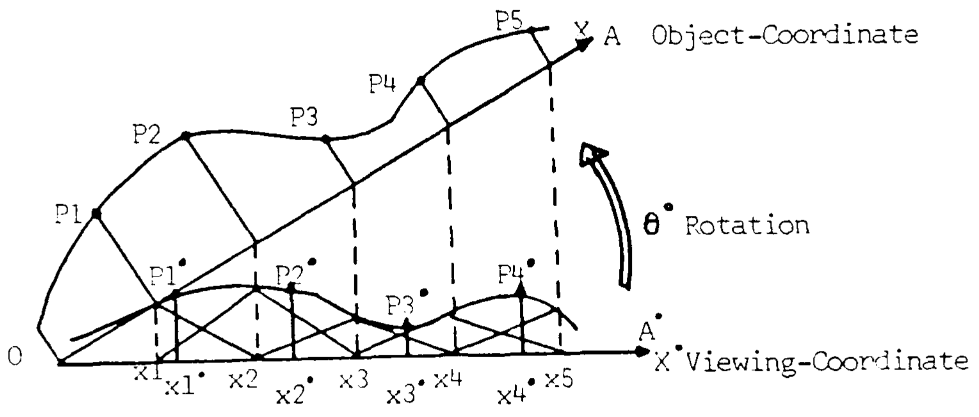


Figure 3.6 Projection Process after Rotation in One-Dimension.

Suppose an arbitrary function is rotated in the counter-clockwise direction by an amount θ degrees. In this case the sample points (P1,P2,P3,•••) will be projected onto the viewing-plane (O-A•). A projected viewing-coordinate value at the grid of viewing-plane will have nonintegers. In the above figure, x1,x2,x3,••• are projected onto viewing-coordinate values with the sample values (P1,P2,P3,•••). Therefore we have to determine the intensity values at the the grids of object-coordinate. Because the adjacent two sample values have a linear relationship, we can determine the intermediate values from these two adjacent sample points (Figure 3.6)

$$P1\bullet = P1 * (x2 - x1\bullet) + P2 * (x1\bullet - x1) \quad (3.2)$$

$$P2\bullet = P2 * (x3 - x2\bullet) + P3 * (x2\bullet - x2) \quad (3.3)$$

B. SINGLE PLANE DISSECTION

The dissection is the technique that an object plane is clearly displayed showing spatial relationship with other planes in the volume data. But only single plane dissection is considered in this work because the dissection process is very similar to the single plane dissection. Single plane dissection is performed by removing all other planes and displaying an object plane.

Suppose we want to see a plane to the selected viewing-angle after rotation. An viewing-plane will be lying in the

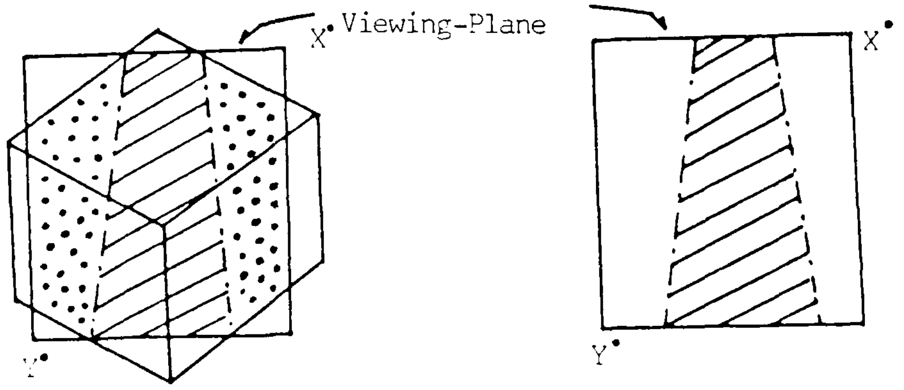


Figure 4.4 . Single Plane Dissection.

Viewing-coordinate shown as in Figure 4.4. After rotation, viewing-plane cuts the volume with an arbitrary angle. The voxels reprojected onto the viewing-plane are not squares rather polygons with an arbitrary shape. Therefore it is difficult to determine what portion of sample values will be contributed to the pixels in the viewing-plane.

To calculate the intensity value of pixels in the viewing-plane, 3-dimansional interpolation is necessary. The interpolation using sphere-shape kernel is used to calculate the intensity values on the viewing-coordinate grids. Subroutine 'Dissection" in the program 'Rot_3_Dim' shows the dissection process. Procedure for this scheme is

(1) Accept a desired rotation angle and desired Z•-coordinate to see.

(2) Take a voxel from the data.

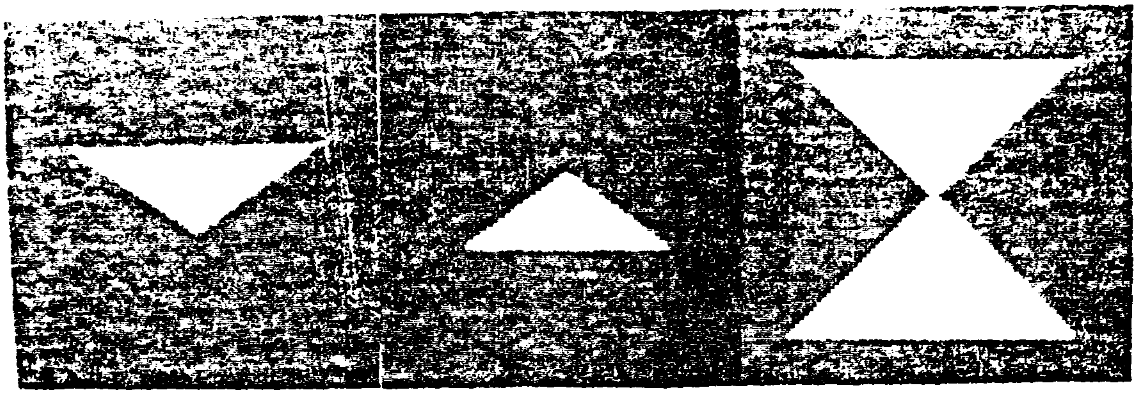
(3) Transform it into the corresponding viewing-coordinate value.

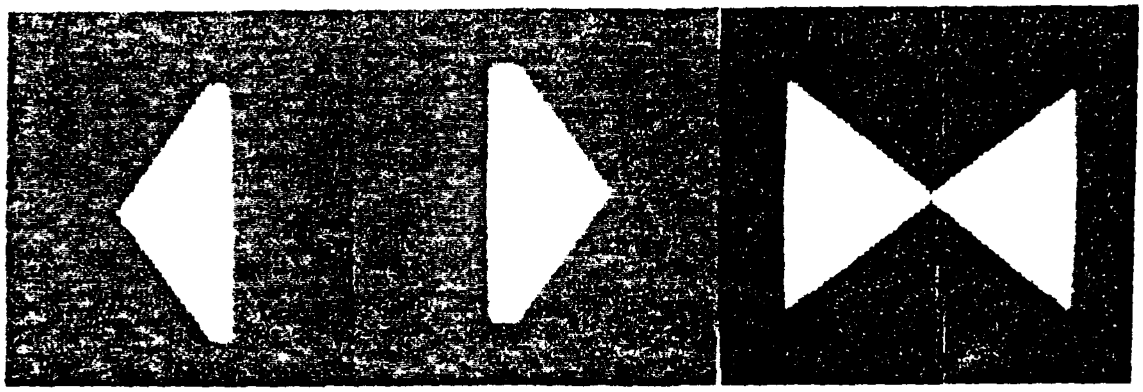
(4) Devide the voxel value to the surrounding 8 coordinate-grids by 3-dimensional interpolation method.

(5) If the Z•-coordinate of the grid corresponds to the desired Z•-coordinate, add the portion of the voxel value to the corresponding pixel in the viewing-plane. Otherwise ignore it.

(6) Continue to the last voxel.

Several dissected planes from the reprojected image are illustrated in Figure 4.5. We can see the checker-board effect in some displayed images which does not appear in the figure. This means that the linear interpolation with sphere-shape kernel is not sufficient for the single plane dissection in the certain rotation angle. If one want higher quality images, more precise and efficient interpolation method should be developed.


45° (ZCRD=25) 60° (ZCRD=-25) 90° (ZCRD=0)
(A) Rotation about X-Axis


45° (ZCRD=20) 135° (ZCRD=20) 80° (ZCRD=0)
(B) Rotation about Y-Axis

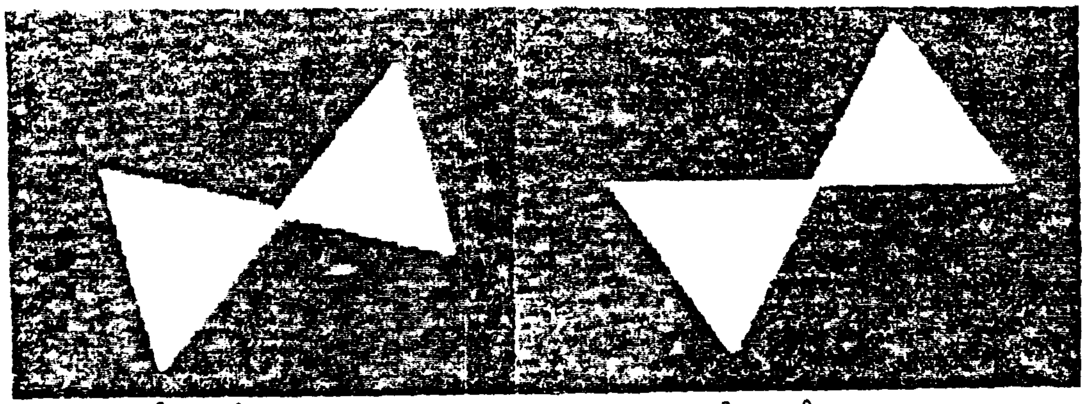

30° 60° (ZCRD=0) 45°-45° (ZCRD=0)
(C) Rotation about Both-Axis

Figure 4.5 Displayed Image by Dissection (ZCRD is the Z'th Slice of Volume Image).

V. CONCLUSION AND RECOMMENDATIONS

A. CONCLUSION

An inherent disadvantage of the reprojection method is the obscuring of underlying and overlying structures in the 2-dimensional viewing-screen. Therefore it is desirable to have better visibility of selected portions of the volume data. Interactive user selection of an adequate viewing-angle and the application of the dissection method is proposed to resolve this problem. Another important problem is the need to manipulate a huge amount of data as efficiently as possible.

Three different algorithms were studied here for the coordinate transformation: Direct matrix multiplication, Decomposition with fixed coordinate, and Recalculation and Indexing.

In the Precalculation and Indexing method, the components of coordinate and matrix element are calculated iteratively and stored in an array. Then these components are indexed according to the incoming coordinates and used in the calculation so that the number of matrix multiplication is decreased. This method requires an appropriate size of memory. It is possible to rotate 64^3 data elements in 25 seconds with the VAX-11/750 minicomputer.

To realize this algorithm in hardware, two hardware design schemes have been suggested in Chapter II.

After rotation to a selected direction, the positions of the voxels do not correspond to the viewing-coordinate grids. Therefore several linear interpolation algorithms are studied.

The reason why linear interpolation using a cone-shape kernel can be extended from 2D to 3D is attributed to its invariant shape with respect to the different directions.

The sphere-shape kernel is a 3D extension of the cone-shape kernel in the 2D situations.

Using this 3D interpolation method, a reprojection and single plane dissection program has been tested with an artificial data. Although linear interpolation using sphere-shape kernel has inherent errors, this method provides the reasonable quality images in a reasonable amount of time.

Manipulation of a huge amount of data such as 3D computed tomography data requires a continuous trade-off between processing time and memory, or between processing time and image quality. The only way to resolve this kind of problems is to use heuristic approaches. First, select an initital algorithm and test it. If the result is not adequate for the specific purpose, try another algorithm. This procedure is continued until an appropriate result can be reached.

B. RECOMMENDATIONS

The coordinate rotation and reprojection method can be widely used in the viewing or manipulating of 3-dimensional data. The algorithms which have been developed in this work may not be optimum because these are the result of heuristic approaches.

The coordinate transformation algorithm implemented here can rotate any reasonable size of 3D data in a short time. But, further studies are required in order to get better quality images from a 3D interpolation algorithm.

It is worth while to test the implementation of the reprojection algorithm with real CT scan data. In this case new problems may arise, which will require further studies.

The implementation of the dissection and the dissolution capabilites are very helpful for the investigation of structures in 3D volume data.

APPENDIX A

PROGRAM OF REPROJECTION AND DISSECTION

```
program rotation_3_dim(infile,input,output);
(*******************************************************************************
                    computer system: VAX-11/750
                    language: VAX-11 PASCAL 2.5

   This program rotates a given 3-dimensional object to the selected direction
with a desired amount of angle and projects it onto the viewing-plane
orthogonal to z_axis by trimetric-projection method, and then displays the
projected image on the COMTAL image display system. Depending on the selection
of projection or dissection, single plane or projected image is displayed.
linear interpolating method using sphere shaped kernel is used in 3D.
*******************************************************************************)

    const
        convert = 3.14159 / 180.0;
        maxcd = 31;
        mincd = -32;
        dimnsn = 64 - 1;
    type
        byte = 0..255;
        angle = 0..180;
        onerec = packed array[0..4095] of byte;
        outimg = packed array[-55..55,-55..55] of real;
        buf = packed array[0..127,0..127] of byte;
        holdmults = packed array[mincd..maxcd] of real;
    var
        infile : file of onerec;(* incoming data file               *)
        angx,angy : angle;      (* rotation angle in degree         *)
        angx,angy : angle;      (* rotation angle in degree         *)
        viewpln : outimg;       (* image array for the projection *)
        outbuf : buf;           (* output array for displaying      *)
        cx,cy,cz:integer;    (* object coord. along x,y,z axis      *)
```

```
     vx,vy,vz : real;    (* viewing coord. along X,Y,Z_axis    *)
     c11x,c12y,c13z : holdmults; (* hold the interminate values for *)
     c21x,c22y,c23z : holdmults; (* calculating view coordinate.    *)
     c31x,c32y,c33z : holdmults; (*                                 *)
     c11,c12,c13,c21,c22: real; (* elements of transformation  *)
     c23,c31,c32,c33: real;     (* matrix                      *)
     ax,ay,az: integer;  (* temperary counter variable *)
     count: integer;     (* temporary counting variable *)
     pixel: integer;     (* incomming intensity value of a pixel *)
     maxval: real;   (* maximum intensity value in the projection plane *)
     z_crd: integer; (* object Z_plane for dissection *)
     want_proj: boolean;(* call projection or dissection subroutine
                           according to the selection             *)
     map : char;     (* select projection or dissection method *)
     dir : char;     (* a selected direction of rotation *)
     tstart,tmtaken : integer;(* CPU time check variables *)

    value
      (* initialize working space & display buffer *)
       viewpln := (111 of(111 of 0.0));
       outbuf := (128 of(128 of 0));

       function trun(num:real):integer;
(******************************************************************************
          Accepts a real number and returns a integer number orienting to
       the negative infinite
******************************************************************************)
          var
             temp : integer; (* temporary storage variable *)
          begin
             if num<0.0 then begin
                 temp := trunc(num);
                 trun := temp;
                 if temp<>num then
                     trun := temp - 1
             end
          else trun := trunc(num)
        end;

      procedure projection(vx,vy,vz: real; pixel: integer);
(*****************************************************************************
         This routine accepts the real viewing coordinates and intensity
      value of each pixel, and then projects this value onto the viewing-
      plane orthogonal to the Z_axis. To interpolate along the object
      coordinate, trilinear interpolation method using sphere shaped kernel
      is used.
*****************************************************************************)
        var
          dxsq1,dxsq2: real; (* squared distance in X_axis *)
          dysq1,dysq2: real; (*    "        "        Y_axis *)
          dzsq1,dzsq2: real; (*    "        "        Z_axis *)
          comp: real;      (* constant for comparing the distance *)
          ix,iy,iz : integer;(* integer coord. to the given real coord.*)

        begin
          ix := trun(vx);
          iy := trun(vy);
          iz := trun(vz);

          dxsq1 := sqr(vx-ix);
          dxsq2 := sqr(1-(vx-ix));
          dysq1 := sqr(vy-iy);
          dysq2 := sqr(1-(vy-iy));
          dzsq1 := sqr(vz-iz);
          dzsq2 := sqr(1-(vz-iz));
          comp := sqrt(dxsq1+dysq1+dzsq1);
          if comp <= 1.0 then
              viewpln[ix,iy] :=(1-comp)*pixel+viewpln[ix,iy];
          comp := sqrt(dxsq2+dysq1+dzsq1);
          if comp <= 1.0 then
              viewpln[ix+1,iy]:=(1-comp)*pixel+viewpln[ix+1,iy];
          comp := sqrt(dxsq1+dysq2+dzsq1);
```

```
            if comp <= 1.0 then
                viewpln[ix,iy+1] :=(1-comp)*pixel+viewpln[ix,iy+1];
            comp := sqrt(dxsq2+dysq2+dzsq1);
            if comp <= 1.0 then
                viewpln[ix+1,iy+1] :=(1-comp)*pixel+viewpln[ix+1,iy+1];

            comp := sqrt(dxsq1+dysq1+dzsq2);
            if comp <= 1.0 then
                viewpln[ix,iy] :=(1-comp)*pixel+viewpln[ix,iy];
            comp := sqrt(dxsq2+dysq1+dzsq2);
            if comp <= 1.0 then
                viewpln[ix+1,iy]:=(1-comp)*pixel+viewpln[ix+1,iy];
            comp := sqrt(dxsq1+dysq2+dzsq2);
            if comp <= 1.0 then
                viewpln[ix,iy+1] :=(1-comp)*pixel+viewpln[ix,iy+1];
            comp := sqrt(dxsq2+dysq2+dzsq2);
            if comp <= 1.0 then
                viewpln[ix+1,iy+1] :=(1-comp)*pixel+viewpln[ix+1,iy+1]
        end;

    procedure dissection(vx,vy,vz: real; pixel: integer);
(*****************************************************************************
        This routine accepts real viewing coordinates and intensity value of
    each pixel, and then extract a desired viewing_plane by projecting the
    intensity value onto the viewing-plane orthogonal to the z_axis.
    Every pixel value consisting viewing-plane is interpolated with
    trilinear interpolation method using sphere shaped kernel.
*****************************************************************************)
        var
            dxsq1,dxsq2: real; (* squared distance in the X_axis *)
            dysq1,dysq2: real; (*    "          "          Y_axis *)
            dzsq1,dzsq2: real; (*    "          "          Z_axis *)
            comp : real; (* temporary value for comparing the distance *)
            ix,iy,iz : integer;(* integer coord. to the given real coord.*)
        begin
            ix := trun(vx);
```

```
iy := trun(vy);
iz := trun(vz);

dxsq1 := sqr(vx-ix);
dxsq2 := sqr(1.0-(vx-ix));
dysq1 := sqr(vy-iy);
dysq2 := sqr(1.0-(vy-iy));
dzsq1 := sqr(vz-iz);
dzsq2 := sqr(1.0-(vz-iz));
if iz = z_crd then begin
  comp := sqrt(dxsq1+dysq1+dzsq1);
  if comp <= 1.0 then
    viewpln[ix,iy] :=(1.0-comp)*pixel+viewpln[ix,iy];
  comp := sqrt(dxsq2+dysq1+dzsq1);
  if comp <= 1.0 then
    viewpln[ix+1,iy]:=(1.0-comp)*pixel+viewpln[ix+1,iy];
  comp := sqrt(dxsq1+dysq2+dzsq1);
  if comp <= 1.0 then
    viewpln[ix,iy+1] :=(1.0-comp)*pixel+viewpln[ix,iy+1];
  comp := sqrt(dxsq2+dysq2+dzsq1);
  if comp <= 1.0 then
    viewpln[ix+1,iy+1] :=(1.0-comp)*pixel+viewpln[ix+1,iy+1]
end;
if iz+1 = z_crd then begin
  comp := sqrt(dxsq1+dysq1+dzsq2);
  if comp <= 1.0 then
    viewpln[ix,iy] :=(1.0-comp)*pixel+viewpln[ix,iy];
  comp := sqrt(dxsq2+dysq1+dzsq2);
  if comp <= 1.0 then
    viewpln[ix+1,iy]:=(1.0-comp)*pixel+viewpln[ix+1,iy];
  comp := sqrt(dxsq1+dysq2+dzsq2);
  if comp <= 1.0 then
    viewpln[ix,iy+1] :=(1.0-comp)*pixel+viewpln[ix,iy+1];
  comp := sqrt(dxsq2+dysq2+dzsq2);
  if comp <= 1.0 then
    viewpln[ix+1,iy+1] :=(1.0-comp)*pixel+viewpln[ix+1,iy+1]
```

```
                end
            end;

        (* FORTRAN program for displaying image on the COMTAL.*)
        procedure displ28(outbuf:buf);fortran;
(***********************************************************************
*                            main program                              *
***********************************************************************)
    begin
        open(infile,'prm.dat',history:=old,access_method:=sequential,
                record_length:=4096,record_type:=fixed);
        reset(infile);

        (* get a desired direction of rotation *)
        writeln('About which axis do you want to rotate("x","y","b")');
        readln(dir);

        (* get a projection method *)
        writeln('Do you want projection or dissection ?("p","d")');
        readln(map);
        if map = 'p' then want_proj := true
        else begin
            want_proj := false;
            writeln('Which plane do you want to see after rotation(-55..55)');
            readln(z_crd)
        end;

        (* branch to 3 directions of rotation. The center of rotation is the
           the center of volume *)
        tstart := clock;
        case dir of
            'x' : begin
                writeln('how much angle in degree(0..180)');
                readln(angx); (* get rotation angle *)
                radx := angx * convert; (* convert degree into radian *)
                c22 := cos(radx);     c23 := - sin(radx);
```

```
          c32 := sin(radx);  c33 := cos(radx);

          c22y[mincd] := mincd * c22; c23z[mincd] := mincd * c23;
          c32y[mincd] := mincd * c32; c33z[mincd] := mincd * c33;
          for count := mincd+1 to maxcd do begin
              c22y[count] := c22y[count-1] + c22;
              c23z[count] := c23z[count-1] + c23;
              c32y[count] := c32y[count-1] + c32;
              c33z[count] := c33z[count-1] + c33
          end;(* end of for *)

          cz := mincd;
          while not EOF(infile) do begin
              for cy := mincd to maxcd do begin
                  vy := c22y[cy] + c23z[cz];
                  vz := c32y[cy] + c33z[cz];
                  for cx := mincd to maxcd do begin
                      vx := cx;
                      pixel := infile^[(cx+32)+(cy+32)*64];
                      if want_proj = true then projection(vx,vy,vz,pixel)
                          else dissection(vx,vy,vz,pixel)
              end (* end of for *)
          end;
          cz := cz + 1;
          get(infile)
      end (* end of while *)
  end;(* end of selection *)

  'y' : begin
      writeln('how much angle in degree(0..180)');
      readln(angy); (* get rotation angle *)
      rady := angy * convert; (* convert degree into radian *)

      c11 := cos(rady);   c13 := sin(rady);
      c31 := - sin(rady);   c33 := cos(rady);
      c11x[mincd] := mincd * c11; c13z[mincd] := mincd * c13;
```

```
   c31x[mincd] := mincd * c31; c33z[mincd] := mincd * c33;
   for count := mincd+1 to maxcd do begin
       c11x[count] := c11x[count-1] + c11;
       c13z[count] := c13z[count-1] + c13;
       c31x[count] := c31x[count-1] + c31;
       c33z[count] := c33z[count-1] + c33
    end;

    cz := mincd;
    while not EOF(infile) do begin
        for cx := mincd to maxcd do begin
             vx := c11x[cx] + c13z[cz];
             vz := c31x[cx] + c33z[cz];
             for cy := mincd to maxcd do begin
                  vy := cy;
                  pixel:=infile^[(cx+32)+(cy+32)*64];
                  if want_proj = true then projection(vx,vy,vz,pixel)
                       else dissection(vx,vy,vz,pixel)
             end (* end of for *)
        end; (* end of for *)
        cz := cz + 1;
        get(infile)
   end (* end of while *)
end;(* end of selection *)

(* this paragraph rotates image along x_axis first, and then
does along y_axis. *)
'b':   begin
        writeln('how much angle in degree(0..180) x=,y=');
        readln(angx,angy); (* get rotation angle *)
        radx := angx * convert; (* convert degree into radian *)
        rady := angy * convert;
        c11 := cos(rady);   c13 :=  sin(rady);
        c21 := sin(radx)*sin(rady); c22 := cos(radx);
        c23 := -sin(radx)*cos(rady); c31 := - cos(radx)*sin(rady);
        c32 :=  sin(radx);  c33 := cos(radx)*cos(rady);
```

```
c11x[mincd] := mincd * c11; c13z[mincd] := mincd * c13;
c21x[mincd] := mincd * c21; c22y[mincd] := mincd * c22;
c23z[mincd] := mincd * c23; c31x[mincd] := mincd * c31;
c32y[mincd] := mincd * c32; c33z[mincd] := mincd * c33;
for count := mincd+1 to maxcd do begin
    c11x[count] := c11x[count-1] + c11;
    c13z[count] := c13z[count-1] + c13;
    c21x[count] := c21x[count-1] + c21;
    c22y[count] := c22y[count-1] + c22;
    c23z[count] := c23z[count-1] + c23;
    c31x[count] := c31x[count-1] + c31;
    c32y[count] := c32y[count-1] + c32;
    c33z[count] := c33z[count-1] + c33
end;

cz := mincd;
while not EOF(infile) do begin
     for cy := mincd to maxcd do begin
          for cx := mincd to maxcd do begin
               vx := c11x[cx] + c13z[cz];
               vy := c21x[cx] + c22y[cy] + c23z[cz];
               vz := c31x[cx] + c32y[cy] + c33z[cz];
               pixel:=infile^[(cx+32)+(cy+32)*64];
               if want_proj = true then projection(vx,vy,vz,pixel)
                   else dissection(vx,vy,vz,pixel)
          end (* end of for *)
       end;
       cz := cz + 1;
       get(infile)
    end; (* end of while *)
  end(* end of selection *)
end;(* end of case *)
close(infile);

(* COMTAL displays only the byte data, so this block find the biggest
   pixel in the projection plane. *)
```

```
          for cx := 17 to 109 do
              outbuf[cy,cx] := round(viewpln[cx-63,cy-63]*255.0/maxval);

      tmtaken := clock-tstart;
      (* place the original image on the center of viewing window *)
      for cy := 32 to 95 do
          for cx := 32 to 95 do
              obuf[cy,cx]:= round(testval[cx-64,cy-64]);

      writeln('tmtaken:':7,tmtaken:6);
      (*  display image on the COMTAL.*)
      disp256(outbuf,1);
      disp256(obuf,2)
end. (*--- end of program--- *)
```

```
for cz := mincd to maxcd do
    for cx := mincd to maxcd do begin
        vx := c11x[cx] + c13z[cz];
        vz := c31x[cx] + c33z[cz];

        ix := trun(vx);
        iz := trun(vz);

        dxsq1 := sqr(vx-ix); .
        dxsq2 := sqr(1-(vx-ix));
        dzsq1 := sqr(vz-iz);
        dzsq2 := sqr(1-(vz-iz));
        comp := sqrt(dxsq1+dzsq1);
        if comp <= 1.0 then
            viewpln[ix,iz] :=(1-comp)*testval[cx,cz]+viewpln[ix,iz];
        comp := sqrt(dxsq2+dzsq1);
        if comp <= 1.0 then
            viewpln[ix+1,iz]:=(1-comp)*testval[cx,cz]+viewpln[ix+1,iz];
        comp := sqrt(dxsq1+dzsq2);
        if comp <= 1.0 then
            viewpln[ix,iz+1] :=(1-comp)*testval[cx,cz]+viewpln[ix,iz+1];
        comp := sqrt(dxsq2+dzsq2);
        if comp <= 1.0 then
            viewpln[ix+1,iz+1] :=(1-comp)*testval[cx,cz]+viewpln[ix+1,iz+1]
    end; (* end of for *)

(* COMTAL displays only a byte data, so this block find the biggest
   value in the viewing-plane  *)
maxval := viewpln[minbf,minbf];
for cy := minbf+1 to maxbf do
    for cx := minbf+1 to maxbf do
        if viewpln[cx,cy] > maxval then maxval := viewpln[cx,cy];

(* rescale the image value in the viewing-plane and place the rotated
   image on the center of viewing-window *)
for cy := 17 to 109 do
```

```
                     trun := temp -1
                 end
                 else trun := trunc(num)
               end;

            (* Fortran program for displaying image on COMTAL.*)
            procedure disp256(obuf:buf: ind:byte);fortran;

(***************************************************************************)
(*     start main program     *)
(***************************************************************************)
        begin
            for cz := mincd to maxcd do
                for cx := mincd to maxcd do
                    testval[cx,cz] := 127*cos(cx + cz) + 127;

            (*  get desired rotation angle *)
            writeln('how much angle in degree(0..180)');
            readln(angy); (* get rotation angle *)
            tstart := clock;
            rady := angy * convert; (* convert degree into radian *)

            (* calculate the coefficient of linear equation *)
            c11 :=  cos(rady);    c13 :=  sin(rady);
            c31 := -sin(rady);    c33 := cos(rady);

            (* calculate  components of equation and store them in array *)
            c11x[mincd] := mincd * c11;  c13z[mincd] := mincd * c13;
            c31x[mincd] := mincd * c31;  c33z[mincd] := mincd * c33;
            for count := mincd+1 to maxcd do begin
                c11x[count] := c11x[count-1] + c11;
                c13z[count] := c13z[count-1] + c13;
                c31x[count] := c31x[count-1] + c31;
                c33z[count] := c33z[count-1] + c33
            end;
        obuf : buf;   (* display buffer to store original image *)
        angy : angle:         (* rotation angle in degree *)
        rady : real;          (* rotation angle in radian *)
        vx,vy,vz : real;      (* viewing-coord. value along x,y,z axis *)
        ix,iy,iz : integer;  (* integer value of viewing-coord. *)
        cx,cy,cz :integer;   (* object-coord. along x,y,z axis *)
        valuezmi,valuezpl: real;(* intermediate value for calculation     *)
        c11x,c12y,c13z:storecom;(* array for store components of mults    *)
        c21x,c22y,c23z:storecom;(*    coord. and matrix-element           *)
        c31x,c32y,c33z:storecom;
        c11,c12,c13 : real;     (* elements of transformation matrix      *)
        c21,c22,c23 : real;
        c31,c32,c33 : real;
        dxsq1,dxsq2,dzsq1,dzsq2: real;(* squared value of x,y,z components *)
        maxval : real;          (* maximum pixel value *)
        comp : real;            (* temporary value variable to comparision  *)
        count : integer;        (* temperary counter variable *)
        tstart,tmtaken : integer;(* time check variable *)

value   (* initialize working space & display buffer *)
        viewpln := (93 of (93 of 0.0));
        outbuf := (128 of (128 of 0));
        obuf := (128 of (128 of 0));

        function trun(num:real):integer;
(**************************************************************************
          this function accepts a real value and returns a integer
        value truncated toward negative infinite.
 *************************************************************************)
          var
              temp : integer;
          begin
              if num < 0.0 then begin
                  temp := trunc(num);
                  trun := temp;
                  if num <> temp then
```

APPENDIX D

PROGRAM OF 2D INTERPOLATION USING CONE-SHAPE KERNEL

```
program intpol_cone_kernel(input,output);
(***************************************************************************
   This program is written for the test of 2D linear interpolation. The
test data is a cosine wave the frequency of which is varing radially.
 Two dimensional cone shaped kernel along the object-coordinate is used for
interpolation. If the distance from the center of a pixel to a viewing-
coordinate grid is less than 1, assign the distribution of a pixel value
to the grids of viewing-coordinate inversely proportional to the distance.
Otherwise assign zero value. Pixel values of viewing-plane resulting from
 the sum of distribution is rescaled into 1 byte range for displaying on
COMTAL image display system.
***************************************************************************)
   const
        convert = 3.14159 / 180.0;
        maxcd =  31;
        mincd = -32;
        maxbf =  46;
        minbf = -46;
        screensiz = 128;
   type
        byte = 0..255;
        angle = 0..180;
        image =  array[mincd..maxcd,mincd..maxcd] of real;
        storecom =  array[mincd..maxcd] of real;
        temparray = array[minbf..maxbf,minbf..maxbf] of real;
        buf = packed array[1..screensiz,1..screensiz] of byte;
   var
        viewpln: temparray;(*output array to store the resulting pixel value*)
        testval: image;    (* array to store the original data *)
        outbuf : buf; (* display buffer to store the rotated image *)
```

```
      for cx := minbf to maxbf do begin
          vx := c11x[cx] + c13z[cz];
          vz := c31x[cx] + c33z[cz];
          if (vx < 31.0) and (vx > -32.0) and
             (vz < 31.0) and (vz > -32.0) then begin
              ix := trun(vx);
              iz := trun(vz);
              valuezmi := (1-(vx-ix))*testval[ix,iz] +
                          (vx-ix)*testval[ix+1,iz];
              valuezpl := (1-(vx-ix))*testval[ix,iz+1] +
                          (vx-ix)*testval[ix+1,iz+1];
              viewpln[cx,cz] := (1-(vz-iz))*valuezmi + (vz-iz)*valuezpl
           end (* end of if *)
      end; (* end of for *)

  (* COMTAL displays only a byte data, so this block find the biggest
     value in the viewing-plane  *)
  maxval := viewpln[minbf,minbf];
  for cy := minbf+1 to maxbf do
      for cx := minbf+1 to maxbf do
          if viewpln[cx,cy] > maxval then maxval := viewpln[cx,cy];
  (* rescale the image value in the viewing-plane and place the rotated
     image on the center of viewing-window *)
  for cy := 17 to 109 do
      for cx := 17 to 109 do
          outbuf[cy,cx] := round(viewpln[cx-63,cy-63]*255.0/maxval);
  tmtaken := clock-tstart;
  (* place the original image.on the center of viewing window *)
  for cy := 32 to 95 do
      for cx := 32 to 95 do
          obuf[cy,cx]:= round(testval[cx-64,cy-64]);
  writeln('tmtaken:':7,tmtaken:8);
  (*  display image on the COMTAL.*)
  disp256(outbuf,1);
  disp256(obuf,2)
end. (*--- end of program--- *)
```

```
                end
                else trun := trunc(num)
            end;

        (* Fortran program for displaying image on COMTAL.*)
        procedure disp256(obuf:buf; ind:byte);fortran;
(****************************************************************************)
(*    start main program    *)
(****************************************************************************)
    begin
        for cz := mincd to maxcd do
            for cx := mincd to maxcd do
                testval[cx,cz] := 127*cos(cx + cz) + 127;

        (*  get desired rotation angle *)
        writeln('how much anile in degree(0..130)');
        readln(angy); (* get rotation angle *)
        tstart := clock;
        rady := angy * convert; (* convert degree into radian *)

        (* calculate the coefficient of linear equation. In this calculation,
        inverse rotation angle is applied. *)
        c11 := cos(rady);   c13 := -sin(rady);
        c31 := sin(rady);   c33 := cos(rady);

        (* calculate  components of equation and store them in array *)
        c11x[minbf] := minbf * c11; c13z[minbf] := minbf * c13;
        c31x[minbf] := minbf * c31; c33z[minbf] := minbf * c33;
        for count := minbf+1 to maxbf do begin
            c11x[count] := c11x[count-1] + c11;
            c13z[count] := c13z[count-1] + c13;
            c31x[count] := c31x[count-1] + c31;
            c33z[count] := c33z[count-1] + c33
        end;

        for cz := minbf to maxbf do
        obuf : buf;          (* display buffer to store original image *)
        angy : angle;        (* rotation angle in degree *)
        rady : real;         (* rotation angle in radian *)
        valuezmi,valuezpl : real; (* temporary value variable *)
        vx,vy,vz : real;       (* viewing-coord. values along x,y,z axis *)
        ix,iy,iz : integer;  (* integer value of viewing-coord. *)
        cx,cy,cz :integer;   (* object coord. along x,y,z axis *)
        c11x,c12y,c13z:storecom;(* array for store components of mults    *)
        c21x,c22y,c23z:storecom;(*    of coord. and matrix-element        *)
        c31x,c32y,c33z:storecom;
        c11,c12,c13 : real;     (* elements of transformation matrix      *)
        c21,c22,c23 : real;
        c31,c32,c33 : real;
        maxval : real;             (* maximum value of pixels    *)
        count : integer;           (* temperary counter variable *)
        tstart,tmtaken : integer;(* time check variable *)

value   (* initialize working space & display buffer *)
        viewpln := (93 of(93 of 0.0));
        obuf := (128 of (128 of 0));
        outbuf := (128 of (128 of 0));

        function trun(num:real):integer;
(**************************************************************************
            this function accepts a real value and returns a integer
        value truncated toward negative infinite.
**************************************************************************)
            var
                temp : integer;
            begin
                if num < 0.0 then begin
                    temp := trunc(num);
                    trun := temp;
                    if temp <> num then
                        trun := temp - 1
```

APPENDIX C

PROGRAM OF 2D INTERPOLATION OF THE SIGNAL FROM THE NEAREST 4 SAMPLES AT THE GRIDS OF THE OBJECT-COORDINATE

```
program intpol_obj_coord(input,output);
(*****************************************************************************
   This program is written for the test of 2D linear interpolation. The
test data is a cosine wave the frequency of which is varing radially.
Two dimensional triangular kernel(direct-rectangular cone) along the
object-coordinate is used. Each viewing-coordinate grid is inversely
transformed into a object-coordinate value, then the pixel value at
the grid of viewing-coordinate is interpolated from 4 surrounding
object-coordinate grids in a bilinear fashion. Pixel values of viewing-
plane resulting from interpolation are rescaled to 1 byte range for
displaying on COMTAL display system.
*****************************************************************************)
   const
        convert = 3.14159 / 180.0;
        maxcd = 31;
        mincd = -32;
        maxbf = 46;
        minbf = -46;
        screensiz = 128;
   type
        byte = 0..255;
        angle = 0..180;
        image =  array[mincd..maxcd,mincd..maxcd] of real;
        storecom =  array[minbf..maxbf] of real;
        temparray = array[minbf..maxbf,minbf..maxbf] of real;
        buf = packed array[1..screensiz,1..screensiz] of byte;
   var
        viewpln: temparray;(*output array to store the resulting pixel value*)
        testval : image;    (* array to store the original data *)
        outbuf :buf;        (* display buffer to store rotated image *)
```

```
          (* get a left lower integer point from real object coord. *)
          ix := trun(vx);
          iz := trun(vz);

          (* distributes a incoming pixel value to surrounding 4 viewing
          coords. *)
          left := (1-(vx-ix)) * testval[cx,cz];
          right := (vx-ix) * testval[cx,cz];
          viewpln[ix,iz+1]  := (vz-iz)*left  + viewpln[ix,iz+1];
          viewpln[ix,iz]    := (1-(vz-iz))*left + viewpln[ix,iz];
          viewpln[ix+1,iz]  := (1-(vz-iz))*right + viewpln[ix+1,iz];
          viewpln[ix+1,iz+1]:= (vz-iz)*right  + viewpln[ix+1,iz+1]
     end; (* end of for *)

(* COMTAL displays only a byte data, so this block find the biggest
   value in the viewing-plane  *)
maxval := viewpln[minbf,minbf];
for cy := minbf+1 to maxbf do
     for cx := minbf+1 to maxbf do
          if viewpln[cx,cy] > maxval then maxval := viewpln[cx,cy];
(* rescale the image value in the viewing-plane and place the rotated
   image on the center of viewing-window *)
for cy := 17 to 109 do
     for cx := 17 to 109 do
          outbuf[cy,cx] := round(viewpln[cx-63,cy-63]*255.0/maxval);
tmtaken := clock-tstart;
(* place the original image on the center of viewing window *)
for cy := 32 to 95 do
     for cx := 32 to 95 do
          obuf[cy,cx]:= round(testval[cx-64,cy-64]);

writeln('tmtaken:':7,tmtaken:6);
(*  display image on the COMTAL.*)
disp256(outbuf,1);
disp256(obuf,2)
and. (*--- end of program--- *)
```

```
          (* Fortran program for displaying image on the COMTAL.*)
          procedure disp256(obuf:buf; ind:byte);fortran;

(****************************************************************************)
(*     start main program     *)
(****************************************************************************)
     begin
          for cz := mincd to maxcd do
               for cx := mincd to maxcd do
                    testval[cx,cz] := 127*cos(cx + cz) + 127;

          (*  get desired rotation angle *)
          writeln('how much angle in degree(0..180)');
          readln(angy); (* get rotation angle *)
          tstart := clock;
          rady := angy * convert; (* convert degree into radian *)

          (* calculate the coefficient of linear equation *)
          c11 := cos(rady);     c13 := sin(rady);
          c31 := -sin(rady);    c33 := cos(rady);

          (* calculate  components of equation and store them in array *)
          c11x[mincd] := mincd * c11;  c13z[mincd] := mincd * c13;
          c31x[mincd] := mincd * c31;  c33z[mincd] := mincd * c33;
          for count := mincd+1 to maxcd do begin
               c11x[count] := c11x[count-1] + c11;
               c13z[count] := c13z[count-1] + c13;
               c31x[count] := c31x[count-1] + c31;
               c33z[count] := c33z[count-1] + c33
          end;

          for cz := mincd to maxcd do
               for cx := mincd to maxcd do begin
                    (* get object coord. *)
                    vx := c11x[cx] + c13z[cz];
                    vz := c31x[cx] + c33z[cz];
```

```
        angy : angle;       (* rotation angle in degree *)
        rady : real;        (* rotation angle in radian *)
        vx,vy,vz : real;    (* viewing-coord. values along x,y,z axis *)
        ix,iy,iz : integer;(* integer value of viewing-coord. *)
        cx,cy,cz :integer; (* object coord. along x,y,z axis *)
        c11x,c12y,c13z:storecom;(* array for store components of mults    *)
        c21x,c22y,c23z:storecom;(*    of coord. and matrix-element        *)
        c31x,c32y,c33z:storecom;
        c11,c12,c13 : real;     (* coefficoents of transformation matrix  *)
        c21,c22,c23 : real;
        c31,c32,c33 : real;
        count : integer;        (* temperary counter variable *)
        maxval : real;          (* maximum value of pixels    *)
        tstart,tmtaken: integer;(* time check variable *)

value   (* initialize working space & display buffer *)
        viewpln := (93 of (93 of 0.0));
        outbuf := (128 of (128 of 0));
        obuf := (128 of (128 of 0));

        function trun(num:real):integer;
(*****************************************************************************
            this function accepts a real value and returns a integer
        value truncated toward negative infinite.
*****************************************************************************)
            var
                temp : integer;
            begin
                if num < 0.0 then begin
                    temp := trunc(num);
                    if num <> temp then
                        trun := temp -1
                end
                else trun := trunc(num)
            end;
```

APPENDIX B

PROGRAM OF 2D INTERPOLATION BY DISTRIBUTING THE SAMPLE VALUES TO THE GRIDS OF VIEWING-COORDINATE.

```
program intpol_view_coord(input,output);
(***************************************************************************
   This program is written for the test of 2D linear interpolation. The
test data is a cosine wave the frequency of which is varing radially.
Two dimensional triangular kernel(direct-rectangular cone) along the
viewing-coordinate is used. An incoming pixel value belongs to each object-
coordinate grid is distributed to surrounding 4 viewing-coordinate grids.
Pixel values of viewing-plane resulting from distribution is rescaled into
1 byte range for displaying on COMTAL.
***************************************************************************)
   const
        convert = 3.14159 / 180.0;
        maxcd = 31;
        mincd = -32;
        maxbf = 46;
        minbf = -46;
        screensiz = 128;
   type
        byte = 0..255;
        angle = 0..180;
        image =  array[mincd..maxcd,mincd..maxcd] of real;
        storecom =  array[mincd..maxcd] of real;
        temparray = array[minbf..maxbf,minbf..maxbf] of real;
        buf = packed array[1..screensiz,1..screensiz] of byte;
   var
        viewpln: temparray;(*output array to store the resulting pixel value*)
        testval: image;     (* array to store the original data *)
        left,right : real; (* intermediate value for calculation *)
        outbuf :buf;        (* display buffer to store the rotated image *)
        obuf : buf;         (* display buffer to store original image   *)
```

```
    maxval := viewpln[-55,-55];
    for ay := -55 to 55 do
         for ax := -55 to 55 do
              if viewpln[ax,ay] > maxval then maxval:=viewpln[ax,ay];

    (*  rescale data values and move it to the output array *)
    for ay := 9 to 119 do
        for ax := 9 to 119 do
            outbuf[ay,ax]:= round(viewpln[ax-64,ay-64]*255.0/maxval);

    tmtaken := clock-tstart;
    writeln('tmtaken:':7,tmtaken:6);

    (*  display image on the COMTAL.*)
    disp128(outbuf)

end. (*--- end of program--- *)
```

LIST OF REFERENCES

1. Herman, G. T., Reynolds, R. A., Udupa, J. K., "Computer Techniques for the Representation of 3D Data on a 2D Display", SPIE, volume 367, Processing and Display of 3D Data, 1982

2. Harris, L. D., Robb, R. A., Yuen, T. S., Ritman, E. L., "Display and Visualization of 3D Reconstructed Anatomic Morphology", Jounal of Computer Assisted Tomography, volume 3, No 4, 1979

3. Keys, R. G., "Cubic Convolution Interpolation for Digital Image Processing", IEEE Transactions on Acoustics, Speech, and Signal Processing, No 6, December 1981

4. Oppenheim, Alan V., Ronald, W. Schafer, Digital Signal Processing, Prentice-Hill, 1975

5. Anton, H., Elementary Linear Algebra, Willey, 1981

6. Roger, Adams, Mathematical Elements for Computer Graphics, McGraw-Hill, 1976

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
2.	Department Chairman, Code 62 Department of Electrical and Computer Engineeing Monterey, California 93943	1
3.	Professor Chin-Hwa Lee, Code 62Le Department of Electrical and Computer Engineeing NAval Postgraduate School Monterey, California 93943	6
3.	Professor Alex Gerba,Jr., Code 62Gz Department of Electrical and Computer Engineeing NAval Postgraduate School Monterey, California 93943	1
4.	Lieutenant Colonel, Hwang, Ho-Sang P.O. Box#17 DaeBang-Dong Dongjak-Gu Seoul, Korea	2
5.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2